UNIVERSIDAD DE CÓRDOBA

INSTITUTO DE ESTUDIOS DE POSTGRADO
PROGRAMA DE DOCTORADO EN INGENIERÍA Y TECNOLOGÍA

OPTIMIZACIÓN DE RECURSOS HARDWARE PARA LA OPERACIÓN DE CONVOLUCIÓN UTILIZADA EN EL PROCESAMIENTO DIGITAL DE SEÑALES

CARLOS DIEGO MORENO MORENO

Departamento de Arquitectura de Computadores, Electrónica y Tecnología Electrónica

LÍNEA DE INVESTIGACIÓN: MODELOS DE SIMULACIÓN Y SU APLICACIÓN A LA RESOLUCIÓN DE SISTEMAS MULTIFÍSICOS EN CIENCIAS, INGENIERÍA Y EDUCACIÓN

MEMORIA DE TESIS PARA OPTAR AL GRADO DE DOCTOR

DIRECTORES:

DR. Dª. PILAR MARTÍNEZ JIMÉNEZ

DR. D. FRANCISCO JOSÉ BELLIDO OUTEIRIÑO

DR. D. FRANCISCO JAVIER HORMIGO AGUILAR

CÓRDOBA, SEPTIEMBRE DE 2013

TITULO: *Optimización de recursos hardware para la operación de convolución utilizada en el procesamiento digital de señales*

AUTOR: *Carlos Diego Moreno Moreno*

Campus de Rabanales
Ctra. Nacional IV, Km. 396 A
14071 Córdoba

www.uco.es/publicaciones
publicaciones@uco.es

Paradoja de la sabiduría:

"Quien sabe mucho, escucha;
quien sabe poco, habla.
Quien sabe mucho, pregunta;
quien sabe poco, sentencia".

Dedicatorias:

A mi madre y a la memoria de mi padre;
A mi mujer Eva;
A mis hijos: mi querido Carlos Diego,
a mi Enrique de mi alma
y a mi pequeña María.

TÍTULO DE LA TESIS:

Optimización de recursos hardware para la operación de convolución utilizada en el procesamiento digital de señales

DOCTORANDO/A:

Carlos Diego Moreno Moreno

INFORME RAZONADO DEL/DE LOS DIRECTOR/ES DE LA TESIS

(se hará mención a la evolución y desarrollo de la tesis, así como a trabajos y publicaciones derivados de la misma).

Dª Pilar Martínez Jiménez, doctora y Catedrática de Universidad Del Departamento de Física Aplicada de la Universidad de Córdoba, D. Francisco José Bellido Outeiriño, doctor y Profesor Colaborador del Departamento de Arquitectura de Computadores, Electrónica y Tecnología Electrónica de la Universidad de Córdoba, y D. Francisco Javier Hormigo Aguilar, doctor y Profesor Titular del Departamento de Arquitectura de Computadores de la Universidad de Málaga, informan que el trabajo titulado "Optimización de recursos hardware para la operación de convolución utilizada en el procesamiento digital de señales" que presenta Carlos Diego Moreno Moreno, Licenciado en Físicas (Electrónica), para optar al grado de Doctor por la Universidad de Córdoba ha sido realizado bajo nuestra dirección.

La presente Tesis Doctoral está bien estructurada y organizada, cubriendo de manera correcta antecedentes, objetivos, desarrollo, conclusiones y bibliografía. La metodología planteada y puesta en práctica demuestra ser la adecuada para resolver los objetivos propuestos.

Establece con claridad la situación actual de las operaciones utilizadas en el procesamiento digital de señales y de la implementación de la operación de convolución en tiempo discreto en los dispositivos electrónicos actuales: FPGAs, DSPs, microcontroladores y microprocesadores; y justifica la necesidad de optimizar los cálculos que se deben realizar en circuitos electrónicos de bajo coste, en particular FPGAs para ejecutar la operación de convolución. Operación matemática que está presente en múltiples facetas del procesamiento digital de señales, tales como el filtrado, correlación o redes neuronales.

La metodología seguida en este trabajo ha comenzado con el estudio de las operaciones utilizadas en el procesamiento digital de señales actualmente tales como la convolución, el filtrado de señales, la correlación o la transformada rápida de Fourier. A continuación se ha realizado un estudio de las publicaciones y antecedentes sobre la convolución y sobre las operaciones de aritmética binaria en general en los dispositivos electrónicos actuales y se ha recopilado la información disponible de los dispositivos electrónicos sobre los que se implementan estas funciones; principalmente FPGAs, DSPs, y GPPs. Se han propuesto nuevas arquitecturas para la implementación de la convolución en

FPGAs, validando y comparando los resultados obtenidos con los de otros dispositivos y de otros autores.

La innovación de esta Tesis Doctoral está basada en la propuesta de nuevas arquitecturas que optimizan los recursos hardware disponibles, de tal modo que los resultados obtenidos en FPGAs de bajo coste igualan o incluso superan a los que se obtienen en FPGAs más potentes. Los resultados del trabajo son directamente aplicables y por tanto, contribuyen no sólo al conocimiento sino al desarrollo tecnológico y a la innovación. Estas arquitecturas incluyen como principal aportación, el uso de la aritmética carry–save en FPGAs para la optimización de la unidad MAC (multiplica-acumula).

En cuanto a la producción científica generada por la Tesis Doctoral destacar el número y calidad de las publicaciones derivadas de ésta, así como la distribución temporal de las mismas.

Publicaciones:

- **Moreno Moreno, Carlos Diego**; Martínez Jiménez, Pilar; Bellido Outeiriño, Francisco José; Hormigo Aguilar, Francisco Javier. "***Comparación de la operación de convolución en diferentes dispositivos electrónicos de bajo coste***". Ponencia en III Congreso Científico de Investigadores en Formación de la Universidad de Córdoba. Córdoba, 9 y 10 de Abril de 2013. Escuela Internacional de Doctorado del Campus de Excelencia Internacional en Agroalimentación (ceiA3, eidA3) y Escuela Multidisciplinar de Doctorado de la UCO (ED-UCO).

- Francisco J. Quiles, Manuel Ortiz, Miguel A. Montijano, **Carlos D. Moreno**, María Brox, Javier Hormigo, Julio Villalba. "*Acelerador Hardware de bajo coste para bus PCI Convencional*". Seminario Anual de Automática, Electrónica Industrial e Instrumentación 2012 (SAAEI'12). Guimarães, Portugal. 11, 12 y 13 de julio de 2012.

- **Moreno, Carlos D.**; Martínez, Pilar; Bellido, Francisco J.; Hormigo, Javier; Ortiz, Manuel A.; Quiles, Francisco J. "***Convolution Computation in FPGA Based on Carry-Save Adders and Circular Buffers***". IT Revolutions. Springer Berlin Heidelberg. D.O.I.: http://dx.doi.org/10.1007/978-3-642-32304-1_20. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Volume 82. Chapter 20. Pages 237-248. ISBN 978-3-642-32303-4. 2012.

- Quiles, F. J.; Ortiz, M.; Brox, M.; **Moreno, C. D.**; Hormigo, J.; Villalba, J. "*UCORE: Reconfigurable Platform for Educational Purposes*". Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on. IEEE Computer Society. Pages 109-114. 13-15 December 2010, Cancun, Quintana Roo, Mexico. D.O.I.: 10.1109/ReConFig.2010.60. ISBN: 978-0-7695-4314-7. INSPEC Accession Number: 11772008.

- **Moreno, C. D.**; Quiles, F. J.; Ortiz, M. A.; Brox, M.; Hormigo, J.; Villalba, J.; Zapata, E. L. "***Efficient mapping on FPGA of convolution computation based on combined CSA–CPA accumulator***". ICECS 2009. 16th IEEE International Conference on Electronics, Circuits, and Systems, 2009. Yasmine Hammamet, Tunisia. Pages 419-422. 13-16 Dec. 2009. D.O.I.: 10.1109/ICECS.2009.5410903. INSPEC Accession Number: 11142053. IEEE Catalog Number: CFP09773. Library of Congress: 2009907318. ISBN: 978-1-4244-5091-6.

- Ortiz, M.; Brox, M.; Quiles, F.; Gersnoviez, A.; **Moreno, C.**; Montijano, M. "Using soft processors for component design in SOC: A case-study of timers". System-on-Chip,

2008. SOC 2008. International Symposium on. IEEE. Pages 113-116. Tampere, Finland. 5-6 Nov. 2008. D.O.I.: 10.1109/ISSOC.2008.4694873. ISBN: 978-1-4244-2541-9. INSPEC Accession Number: 10411531. Citado en WoK, Accession Number:WOS:000262647700031.

Por todo ello, se autoriza la presentación de la tesis doctoral.

Córdoba, 24 de septiembre de 2013

Firma de los directores

Fdo.: Pilar Martínez Jiménez – Fdo.: Francisco J. Bellido Outeiriño – Fdo.: Fco. Javier Hormigo Aguilar

Resumen

Esta tesis presenta varias arquitecturas sobre la unidad MAC (multiplica–acumula) para la optimización de la operación de convolución, que es ampliamente utilizada en el procesamiento digital de señales, sobre varios dispositivos electrónicos de bajo coste. Básicamente esta optimización se centra en las FPGA de Xilinx Spartan 3 y Spartan 6, utilizando aritmética redundante, en particular la aritmética carry–save. Este tipo de aritmética no se suele utilizar en las FPGAs debido a que aumenta el área consumida, pero en esta investigación se ha demostrado experimentalmente que cuando el número de operaciones MAC a realizar es elevado, como es el caso de la convolución de dos señales, el uso de la aritmética CSA resulta eficiente, ya que disminuye significativamente los tiempos empleados, sin un aumento excesivo de los recursos utilizados de la FPGA.

Por otro lado, también se han estudiado otros dispositivos electrónicos que suelen ser empleados en el procesamiento digital de señales, tales como DSP o GPP, realizando una comparación de los tiempos empleados de las FPGAs respecto a estos dispositivos.

Abstract

This Thesis presents several architectures of the multiply-accumulate unit (MAC) to optimize the convolution operation, which is widely used in digital signal processing, on several low-cost electronic devices. This optimization is mainly focused on Xilinx Spartan-3 and Spartan-6 FPGAs, using redundant arithmetic, specifically the carry-save arithmetic (CSA). This type of arithmetic is not usually used on FPGAs since its high consumption of area resources, but this research shows that if the number of MAC operations developed is high, as the case of the convolution of two signals, the use of CSA arithmetic is efficient, since it decreases significantly the execution times without an excessive increase of the resources used in the FPGA.

On the other hand, other electronic devices as DSP or GPP, usually used in digital signal processing, have been studied. A comparation of execution times on FPGAs and these devices has been included.

Agradecimientos

Al comienzo de esta memoria quiero expresar mi agradecimiento a todas las personas que me han ayudado, de forma directa o indirecta, al desarrollo de este trabajo. Particularmente deseo agradecer:

- A los directores de la tesis: Dra. Pilar Martínez Jiménez, Dr. Francisco José Bellido Outeiriño y Dr. Francisco Javier Hormigo Aguilar; por su dedicación y su acertada labor de dirección.

- A mis compañeros de área Francisco Javier Quiles Latorre y Manuel Agustín Ortiz López por su ayuda imprescindible, fundamental e incondicional en la realización de esta tesis.

- Al Departamento de Arquitectura de Computadores de la Universidad de Málaga, en particular al Dr. Julio Villalba Moreno por su colaboración y orientación en las investigaciones realizadas en esta tesis.

- A todos los miembros del departamento de Arquitectura de Computadores, Electrónica y Tecnología Electrónica de la Universidad de Córdoba, en especial a Miguel Ángel Montijano Vizcaíno, María Brox Jiménez, Andrés Gersnoviez Milla, Edmundo Sáez Peña y Antonio Moreno Fernández–Caparrós, por su apoyo y comprensión en todos estos años.

- A mi madre y a la memoria de mi padre, que tanto han insistido en que concluyera este trabajo. A mi familia: a Eva por su comprensión y paciencia, y a mis tres hijos: Carlos Diego, Enrique y María.

A todos, ellos mi más sincera gratitud.

Índice de figuras

Índice de tablas

Índice de contenidos

1 Introducción: objetivos de la tesis

RESUMEN

En este capítulo se expone una breve descripción de la historia del procesamiento digital de señales, hasta llegar a nuestros días. Se comentan las principales aplicaciones de este procesamiento, para justificar la operación de la convolución de dos señales, que es el objetivo principal de esta tesis doctoral. A continuación se describe un breve comentario de los dispositivos electrónicos actuales para concluir que la mejor opción es la realización de esta operación en FPGA, que son dispositivos de bajo coste, muy versátiles y de mejor utilidad para el tratamiento digital de la señal. Concluye el capítulo con la exposición de los objetivos generales de la tesis, el grado de innovación previsto y la estructura de la memoria.

1.1. Introducción

1.1.1. Historia del procesamiento digital de señales

El procesamiento de señales es un campo que tiene sus orígenes en los matemáticos de los siglos XVII y XVIII, y actualmente se ha transformado en una herramienta indispensable en varios campos de la ciencia y la tecnología. Las técnicas y aplicaciones de este campo son tan viejas como Newton y Gauss y tan actuales como las computadoras digitales y los circuitos integrados.

El procesamiento de señales tiene una larga y rica historia. Es una tecnología que se entremezcla con un conjunto de disciplinas amplio entre las que están las telecomunicaciones, el control, la medicina, la exploración del espacio y la arqueología, por citar algunas. Actualmente, esta afirmación es más cierta con la televisión digital, los sistemas de información y los sistemas multimedia. Es más, a medida que los sistemas de comunicación se van convirtiendo cada vez más en sistemas sin hilos, móviles y

multifunción, la importancia del procesamiento de señales en dichos sistemas tiene cada vez más importancia.

Debido a que el procesamiento digital de una señal requiere efectuar ciertos cálculos a partir de los datos disponibles, y que con frecuencia dichos cálculos se pueden realizar de forma manual, podemos afirmar que el procesamiento de señales se practicó durante varios siglos, mucho antes de la aparición de los actuales computadores, en casos tales como el análisis y la predicción del movimiento de cuerpos celestes, o en el análisis y la predicción de las mareas.

El modelo matemático en que se sustenta el estudio de las señales continuas está basado en las transformadas de Fourier y Laplace, desarrolladas a lo largo del siglo XIX. En 1822 Jean Baptiste Joseph, Barón de Fourier, publicó un estudio sobre el flujo calorífico donde desarrolló las series de Fourier que desde entonces han sido aplicadas a diversas ramas de la ciencia, pero principalmente en el análisis de señales. Por otro lado, el astrónomo teórico Pierre Simon, Marqués de Laplace, desarrolló una matemática que fue aplicada al estudio y conocimiento de los planetas.

Posteriormente, en 1928 Harry Nyquist publicó su artículo "*Certain topics in Telegraph Transmission Theory*" [1] en el cual presentaba el efecto producido en el espectro de frecuencia de una señal analógica al ser discretizada en el tiempo, el cual constituyó un gran avance en el procesamiento digital de señales. En este artículo se planteó que, para preservar la información original, la tasa de muestreo debía ser mayor que el doble de la máxima componente de frecuencia contenida en la señal analógica.

Por otro lado, Claude Shannon publicó en 1949 el artículo "*Communications in the Presence of Noise*" [2], donde se demostró que se puede reconstruir perfectamente una señal analógica a partir de sus muestras, si se dispone de un filtro paso–baja analógico ideal. Aunque no es posible la realización de un filtro de este tipo, en muchos casos prácticos se puede realizar una buena aproximación.

El procesamiento de señales estudia la representación, transformación y manipulación de señales y de la utilidad de que disponen. Cuando nos referimos al procesado digital de señales, nos referimos a la representación mediante secuencias de números de precisión finita y el procesamiento se realiza mediante una computadora digital.

Con frecuencia se desea que estos sistemas trabajen en tiempo real, lo cual significa que el sistema en tiempo discreto se implementa de manera que las muestras de salida se van calculando a la misma velocidad con la que se muestrea la señal en tiempo continuo. Son muchas las aplicaciones que requieren esta especificación. El tratamiento en tiempo discreto y en tiempo real de señales en tiempo continuo es práctica común en sistema de control, comunicaciones, radar, sonar, codificación y realce de voz y vídeo, ingeniería biomédica, etc.

Saber interpretar las señales es uno de los problemas de los que se ocupa el tratamiento de señales. Por ejemplo, comprender la señal de entrada es el objetivo principal de un sistema de reconocimiento de la voz humana. Normalmente, en este sistema se aplicará un procesado digital previo (filtrado, estimación de parámetros, etc.) seguido de un sistema de reconocimiento de patrones que obtenga una representación simbólica.

El procesamiento digital de señales ha ido avanzado con pasos desiguales durante un largo periodo de tiempo. Hasta principios de los años cincuenta se realizaba con circuitos electrónicos o con dispositivos mecánicos. Aunque las computadoras digitales ya existían en empresas y en laboratorios científicos, todavía eran económicamente inaccesibles y tenían una capacidad limitada relativamente. Las computadoras digitales se utilizaron para el tratamiento de señales en la prospección petrolífera. Se grababan los datos sísmicos en cintas magnéticas para su posterior procesamiento. Aunque el procesamiento de señales utilizando computadoras digitales tenías grandes ventajas de flexibilidad, sin embargo, era difícil realizarlo en tiempo real.

En 1965 Cooley y Tukey [3], aportaron un algoritmo eficiente para calcular la transformada de Fourier lo cual hizo que se utilizaran más las computadoras digitales. Muchas aplicaciones desarrolladas requerían del análisis espectral de la señal y con las nuevas transformadas rápidas se redujo el tiempo de cálculo. Además, el nuevo algoritmo se podría implementar en un hardware digital específico, por lo que muchos algoritmos de tratamiento digital de señales que anteriormente eran imposibles comenzaron a verse como posibles.

Otro desarrollo fundamental en la historia del Procesamiento de Señales ocurrió en el terreno de la Microelectrónica. Los primeros microprocesadores eran demasiado lentos

para implementar en tiempo real gran parte de los sistemas en tiempo discreto, pero a mediados de los años ochenta la tecnología de los circuitos integrados había avanzado hasta el nivel de permitir la realización de microcomputadores en coma fija y coma flotante con arquitecturas específicas para realizar algoritmos de procesamiento de señales en tiempo discreto. A estos procesadores se les conoce por el nombre de Procesadores Digitales de Señal (DSP, *Digital Signal Processor*). El avance de esta tecnología permitió ampliar las aplicaciones de las técnicas de tratamiento de la señal en tiempo discreto.

El procesamiento de señales es una disciplina que avanza continuamente en sus fundamentos teóricos y en sus aplicaciones. Estas aplicaciones crecen día a día, y son cada vez son más sofisticadas, debido en parte a que las tecnologías asociadas son más accesibles y potentes. En efecto, el progreso alcanzado en términos de velocidad, de capacidad de memoria y programación ponen a disposición de los usuarios algoritmos numéricos en tiempo real para la implementación de todas las técnicas del tratamiento digital de la señal.

El análisis digital de señales es un campo de estudio que se relaciona con el procesamiento de información presentada en forma digital donde, la llegada en los últimos años de modernos sistemas de computación, ha logrado dar un impulso a la introducción de nuevos métodos de análisis que apoyan a los tradicionales, así como un enorme crecimiento en la cantidad de aplicaciones prácticas.

Algunos ejemplos se encuentran en el área de Mecánica Aplicada, donde se utiliza en problemas de tipo estructural, para el diagnóstico de problemas dinámicos en máquinas rotantes, detección de mal funcionamiento de componentes en plantas nucleares; análisis de imágenes acústicas y de sonar marino en problemas de acústica y sonido, así como en reconocimiento y síntesis de voces; en comunicaciones es utilizada en análisis de sistemas y detección de señales, filtrado en canales múltiples, estimación de funciones transferencia, etc.; en ingeniería biomédica permite el monitoreo de la fatiga muscular, investigación en perturbaciones gástricas, en diagnóstico de pacientes con problemas cardíacos y muchas otras aplicaciones dentro de esta área. El procesamiento de señales no está limitado a datos unidimensionales ya que es también aplicable en otra diversidad de aplicaciones tales como procesamiento de imágenes, análisis de señales en un conjunto de antenas, en problemas de trayectorias de transmisión de señales, etc.

El procesamiento digital de señales se ha vuelto un campo significativamente rico en métodos, conocimientos y aplicaciones debido fundamentalmente a la alta tecnología asociada con las computadoras digitales y diseños de equipos digitales (tales como los analizadores de señales, tarjetas de adquisición de datos, grabadores multicanal, colectores de datos, osciloscopios digitales, filtros, etc.), que en combinación con las nuevas técnicas de análisis han generado nuevas vías para llevar adelante estudios de problemas cada vez más complejos.

El tratamiento digital de la señal, como se entiende actualmente, tiene su origen en los años sesenta con la utilización comercial de los primeros computadores digitales. En aquel entonces los sistemas de comunicaciones habían alcanzado una complejidad tal que su diseño y desarrollo, basándose en prototipos, implicaban costes prohibitivos. Como alternativa en las primeras fases de diseño, donde se estudian la viabilidad y las prestaciones básicas de las diferentes posibilidades, se acudió a la simulación mediante computador. Las señales, que se modelaban como funciones de variable real (el tiempo analógico), se representaron por secuencias de muestras, de modo que pasaron a ser funciones de variable entera (el ordinal o el tiempo discreto). De acuerdo con ello, los sistemas (analógicos), que actuaban sobre funciones de variable real, fueron sustituidos por sistemas (discretos) que manejaban secuencias de números. De esas fechas (1967) data el algoritmo *Fast Fourier Transform* (FFT) que permite el cálculo de la transformada de Fourier con un reducido coste computacional. Dada la relativamente escasa capacidad operacional de los computadores de la época, el cálculo de la transformada de Fourier, imprescindible para la descripción de las prestaciones de los sistemas de comunicaciones, era inabordable antes de la aparición de este algoritmo; por ello, se considera que la FFT proporcionó al tratamiento digital de la señal una excelente plataforma de lanzamiento.

Durante la década de los setenta se asiste a un desarrollo continuado de la tecnología digital. En 1972 aparece el primer microprocesador de propósito general y en 1980 el primer microprocesador especializado en el tratamiento de señal (DSP), diseñado para realizar eficientemente el cálculo reiterado de la combinación producto–acumulación (MAC, operación básica de la convolución). Al mismo tiempo, comienza un desarrollo vertiginoso de la teoría fundamental del tratamiento digital de la señal y la exploración de su aplicación práctica en un sinfín de campos. Al convertir la manipulación de las señales en una cuestión de cálculo numérico realizada en un computador, el DSP pudo incorporar

a su patrimonio todos los conocimientos matemáticos o de cualquier otra índole susceptibles de ser programados en un computador. Así, la simulación de sistemas analógicos pronto se convirtió en una más de las muchas tareas que se podían abordar, y tomaron nuevo impulso actividades como el desarrollo de diversos tipos de radar inteligente, el reconocimiento y la síntesis de voz, nuevos sistemas de control, etc. No obstante, la tecnología digital era cara y sus prestaciones modestas, por lo que su aplicación industrial se limitaba a la manipulación de señales con reducido ancho de banda y a productos con una repercusión económica importante: telefonía y aplicaciones militares, sustancialmente.

A partir de 1980 se produce un espectacular avance en las prestaciones de los DSP y en el abaratamiento de sus costes. Hoy en día se encuentran disponibles microprocesadores que realizan en aritmética real más de 10 millones de operaciones MAC por segundo (10 Mflops). Ello ha permitido la proliferación de aplicaciones industriales del tratamiento digital de la señal, que ha impuesto su presencia en campos tan dispares como las comunicaciones, el control, la robótica, la electromedicina, la geofísica e incluso la electrónica de consumo. El DSP ha facilitado, por ejemplo, que hoy pueda disponerse de servicios tales como la telefonía celular, las comunicaciones de datos vía red telefónica o la televisión digital, de sistemas de conversión de texto a voz, o que en nuestros hogares disfrutemos de efectos musicales especiales.

1.1.2. Operación de convolución

La convolución y otras operaciones relacionadas se enmarcan dentro de muchas aplicaciones de ingeniería y matemáticas. Por ejemplo, en estadística, un promedio móvil ponderado se representa mediante una convolución. En óptica, muchos tipos de "manchas" se describen con convoluciones, una sombra es la convolución de la forma de la fuente de luz que crea la sombra y del objeto cuya sombra se está proyectando, una fotografía desenfocada es la convolución de la imagen correcta con el círculo borroso formado por el diafragma del iris. En acústica, un eco es la convolución del sonido original con una función que represente los objetos variados que lo reflejan. En ingeniería eléctrica y otras disciplinas, la salida de un sistema lineal es la convolución de la entrada con la respuesta del sistema a un impulso. En física, allí donde haya un sistema lineal con un "principio de superposición", aparece una operación de convolución. En particular, en el procesamiento digital de señales, la operación de convolución es la clave de muchas

aplicaciones, tales como el filtrado, correlación, redes neuronales, etc. Genéricamente se puede decir, que cuando la variación de una señal influye en la variación de otra, se puede representar matemáticamente esta variación como una convolución de las dos señales.

La operación de convolución en tiempo discreto se reduce en los dispositivos microprogramables a gran cantidad de multiplicaciones y sumas (operación multiplica y acumula MAC), por lo cual resulta imprescindible la investigación de nuevas arquitecturas que implementen estas operaciones aprovechando al máximo los recursos hardware disponibles, y además que realicen estos cálculos a la mayor velocidad posible.

El procesamiento digital de una señal requiere (en muchos casos) de la realización de un gran número de cálculos, haciéndolo inviable si no se dispone de una máquina calculadora de gran velocidad o de un computador. Este problema dificultó el avance en el área de DSP hasta los años 60 y 70, época en la cual progresó rápidamente, gracias a la disponibilidad de grandes computadores (mainframes) en las instituciones. Algunos de los tópicos abordados fueron el diseño e implementación de filtros digitales, invención y optimización del algoritmo de la FFT, compresión de voz, procesamiento de imágenes (fotos tomadas por satélites y naves espaciales) y sismología (búsqueda de minerales y de petróleo).

En esa época las aplicaciones de DSP al procesamiento de señales en tiempo real (tales como radar, sonar, cancelación de ecos, modems) eran muy limitadas. Los procesadores digitales de señal se construían con centenares de circuitos integrados TTL, tenían un costo prohibitivo (excepto para aplicaciones militares) y eran muy complejos. Por lo tanto, la mayoría de los trabajos consistía en desarrollar y ensayar algoritmos en los grandes computadores que poseían las universidades, empresas y otras instituciones.

La capacidad de efectuar multiplicaciones en forma rápida es el requerimiento más importante para poder realizar algoritmos de procesamiento digital de la señal en tiempo real. Los procesadores de esa época no eran capaces de multiplicar en forma directa, sino que lo hacían en base a sumas y desplazamientos.

La multiplicación en hardware requería de una gran área en la pastilla de silicio. A principios de los años 80 se logró reducir el tamaño de los transistores lo suficiente como para poder fabricar un procesador digital de señal capaz de multiplicar 2 números en 1

ciclo de máquina (800 nseg). Con el progreso de la tecnología de integración, este tiempo ha disminuido actualmente a unos pocos nseg.

El desarrollo de conversores A/D y D/A cada vez más rápidos, de mayor resolución (nº de bits), menor tamaño y menor costo, también ha contribuido a reemplazar el procesamiento analógico por el procesamiento digital.

El progreso en la velocidad de cálculo de los microcomputadores personales ha permitido usarlos en tareas cada vez más exigentes, tales como grabación y reproducción de audio y video, procesos que deben efectuarse en tiempo real. Los primeros computadores personales, fabricados a finales de los años 70, sólo podían efectuar algunos centenares o miles de operaciones de punto flotante por segundo (flops). La velocidad de cálculo de los PC actuales ya ha superado los mil millones de flops. Muchos equipos electrónicos complejos se construyen actualmente usando como base un PC industrial, aprovechando el bajo costo del hardware y del software asociado.

1.1.3. Circuitos electrónicos para la operación de convolución

Los dispositivos electrónicos actuales (FPGAs, DSPs, microcontroladores, microprocesadores, etc.) disponen de gran cantidad de recursos internos, tales como FIFOs, BlockRAMs, multiplicadores, lógica rápida, etc., que las hacen ideales para el rendimiento de tareas de procesamiento digital de señales. Sin embargo, algunas de estas aplicaciones requieren tal cantidad de operaciones y a tal velocidad, que exigen el máximo aprovechamiento de los recursos internos de los dispositivos mencionados.

Existen en el mercado una gran variedad de circuitos integrados, que abarcan desde dispositivos personalizados para cada aplicación a otros que se pueden programar en función de las necesidades de cada momento. En la figura 1-1 puede observarse un esquema básico de los diferentes tipos de circuitos integrados.

Figura 1-1 Tipos de circuitos integrados.

Desde el desarrollo de los primeros prototipos de FPGA (*Field Programmable Gate Array*) hace ya tres décadas, estos dispositivos han sufrido importantes avances en cuanto a prestaciones y densidad, al mismo tiempo que los precios han ido disminuyendo. Esta evolución de la tecnología junto con los avances de las herramientas de desarrollo y las tecnologías de soporte han revolucionado el campo del diseño lógico.

A finales de los 70 y principios de los 80 aparecieron un nuevo tipo de circuito llamado ASIC (*Application-Specific Integrated Circuit* – Circuitos Integrados de Aplicación Específica). Los ASIC's más populares eran los *Mask–Programmed Gate Array*. Como su nombre indica, éstos están compuestos de un array o matriz de puertas lógicas. Estos circuitos se programan para una aplicación en particular mediante la creación de máscaras metálicas que determinan la interconexión entre las puertas. No obstante, tienen sus inconvenientes como, por ejemplo, el hecho de que el tiempo desde que se realiza el prototipo hasta que puede ser comercializado es muy elevado. Además, dicho tiempo podría incrementarse ya que, a menudo, los prototipos no funcionan como deben y eso añade tiempo y dinero al desarrollo del producto. Por tanto, el desarrollo de una aplicación basada en un *Gate Array* tiene un considerable riesgo para las empresas.

El estado del arte en el diseño de algoritmos numéricos para ser realizados directamente en circuitos integrados se ha enfocado en los últimos tiempos hacia las FPGA. Soluciones tradicionales como microprocesadores o procesadores digitales de señales DSP no cumplen con las especificaciones de resolución y/o velocidad que requieren algunos procesos digitales para el cómputo en línea de las acciones y resultados de los mismos. Estos procesadores de propósito general tienen la desventaja de estar limitados por la arquitectura cerrada de su diseño a realizar procesamiento secuencial, por

otro lado, los FPGA no tienen esta limitación y pueden ser diseñadas las estructuras digitales con técnicas alternativas para incrementar la velocidad de cómputo y ajustar la resolución requerida, gracias a la reconfigurabilidad de estos dispositivos. Resulta muy importante conocer y manejar estas técnicas de diseño estructural con FPGA para poder plantear soluciones eficientes a los procesos digitales de señales.

Una de las decisiones fundamentales a la hora de diseñar un producto electrónico es la elección del dispositivo electrónico a emplear, ya que ésta va a afectar a variables tales como el coste de dicho producto, sus prestaciones, su método de diseño, etc. Esta decisión debe ser tomada en las primeras etapas del proceso de diseño.

A la hora de elegir entre los diferentes circuitos integrados se deben tener en cuenta diversos aspectos como son la velocidad a la que se desea que trabaje el dispositivo, la cantidad de recursos lógicos que se van a emplear, los costes de diseño y fabricación, el número de unidades a producir o si el dispositivo necesitará ser reprogramado o no.

Así, las FPGA's, inventadas a mediados de los 80, solucionaron muchos de estos problemas. Estos circuitos integrados de alta densidad combinan la flexibilidad y el alto nivel de integración de los anteriores con una sencillez de programación. Como los *Gate Arrays*, las FPGA's contienen una matriz de elementos lógicos que pueden ser interconectados para implementar una aplicación dada. Estas interconexiones son controladas por *switches* programables por el usuario. Así, los prototipos pueden ser implementados, probados y modificados rápidamente.

Un FPGA es un array de bloques lógicos programables colocados en una infraestructura de interconexiones programables; es posible programar la funcionalidad de los bloques lógicos, las interconexiones entre bloques y las conexiones entre entradas y salidas. Un FPGA es programable a nivel hardware. Así, un FPGA proporciona las ventajas de un procesador de propósito general y un circuito especializado que puede reconfigurarse las veces que sea necesario para depurar su funcionalidad. El tamaño y velocidad de los FPGA's son equiparables a los ASIC, pero los FPGA son más flexibles y su ciclo de diseño es más corto.

En resumen, las FPGA's en comparación con los demás dispositivos lógicos, tienen como objetivo destacar en aspectos tales como:

- Velocidad: Aumenta la frecuencia de operación de los sistemas.
- Densidad y capacidad: Al ser sistemas complejos, necesitan de gran cantidad de recursos como puertas lógicas y biestables.
- Facilidad de uso: que permita al ingeniero de desarrollo implementar un diseño con la rapidez que exige el mercado. Esto implica software de desarrollo fácil de usar.
- Programabilidad en el sistema y reprogramabilidad en el circuito: Esto es, la posibilidad de programar o reprogramar el dispositivo que se encuentra ya en el circuito impreso al que va destinado.

El rango de aplicaciones de las FPGAs es muy amplio, debido a la versatilidad y a la flexibilidad de estos dispositivos. La principal aplicación de las FPGAs está orientada al procesamiento digital de señales (DSP), la cual es empleada en comunicaciones, procesado de datos, etc. La elección de una FPGA para aplicaciones de tratamiento de señal se debe a su alta frecuencia de trabajo, a su capacidad de procesamiento en paralelo, y a su bajo precio en comparación con los ASICs.

Las FPGAs y CPLD (Dispositivo Lógico Programable Complejo) son dispositivos semiconductores reprogramables, basados en matrices de bloques de lógica configurables (conocidos como CLBs) cuyas conexiones se pueden modificar según las necesidades del diseñador. En general, la lógica de un CPLD es insuficiente para realizar dicho procesamiento. Esta capacidad de reprogramación, unida a un proceso de desarrollo y fabricación menos costoso y largo que el de otros dispositivos otorga a las FPGAs una posición privilegiada para una gran variedad de aplicaciones frente a otros circuitos integrados. En la actualidad las FPGAs están implantadas en muchos sectores de la industria, tales como procesamiento digital de señales, sistemas militares, prototipado de la funcionalidad de otros dispositivos electrónicos (por ejemplo un ASIC), etc.

Para poder hablar de las características y las ventajas de una FPGA frente a otros dispositivos es necesario describir la arquitectura interna de una FPGA. Básicamente, una FPGA consiste en:

- Una matriz de bloques de lógica configurables (CLB): bloques lógicos cuyos parámetros se pueden modificar de forma que presente diferentes funcionalidades. Cada CLB consiste, de forma genérica, en varias tablas de LUTs (*look–up tables*, o tablas de consulta) cuyas salidas están multiplexadas y unos parámetros de configuración de dicho CLB. El diseño de un CLB varía de un fabricante a otro.

- Una matriz de rutado, encargada de conectar los CLBs entre ellos, y a su vez éstos con las celdas de entrada y salida.

- Celdas de entrada y salida (IOB): su misión es comunicar la lógica interna de la FPGA con el exterior.

No sólo los bloques de lógica son configurables, sino que tanto la matriz de rutado como las celdas de entrada y salida son programables, otorgándole a las FPGAs de una gran flexibilidad a la hora de ajustarse a las especificaciones de cada diseño. (Figura 1-2)

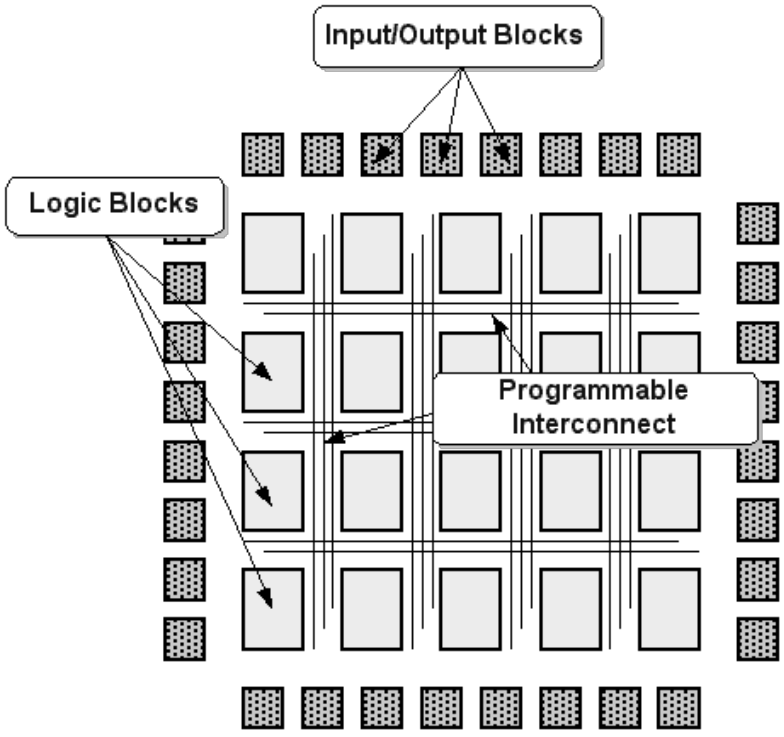


Figura 1-2 Estructura genérica de una FPGA.

Las FPGAs son un producto intermedio entre los dispositivos de lógica programable (PLD), y los circuitos integrados de aplicación específica (ASIC).

Los PLDs surgieron por la necesidad de disponer de circuitos cuya funcionalidad no estuviera definida de forma estática, como un simple conjunto de puertas lógicas conectadas entre sí. Esta funcionalidad se modifica cuando se programa el dispositivo.

Los primeros dispositivos PLD fueron las matrices de lógica programable (PLA), que aparecieron en los años 70. Los PLAs eran chips programables que contenían una serie de puertas AND y OR conectadas por unas matrices de conmutación. Presentaban la desventaja de que podían ser programados una única vez, presentando desde el momento de su programación la misma funcionalidad.

Debido a esto aparecieron las matrices de lógica genérica (GAL), cuya principal diferencia con los PLAs es que pueden ser reprogramadas. Sin embargo, al igual que ocurre con aquellos, las GALs presentan una cantidad muy limitada de puertas lógicas.

Posteriormente aparecieron unos PLDs más complejos, con el nombre de CPLDs, cuyo número de puertas lógicas suele ascender a miles de éstas.

Finalmente, las FPGAs son dispositivos de cientos de miles, e incluso millones, de puertas lógicas. Además de la lógica, también presente en los elementos anteriormente descritos, presentan recursos especiales para implementar de forma eficiente funciones aritméticas (comparadores, sumadores, contadores, etc.), mientras que los CPLD carecen de éstos. Están basadas en memoria RAM, lo cual implica que deben ser configuradas cada vez que se van a utilizar, y necesitan lógica adicional para inicializarlas. Las FPGAs son dispositivos muy flexibles, que pueden trabajar a altas frecuencias y con capacidad de procesamiento en paralelo.

Por su parte, los ASICs son circuitos diseñados de forma particular para cada aplicación. Debido a esta personalización del dispositivo, presentan unos costes fijos mayores que una FPGA, el proceso de fabricación es más largo, etc. Son económicamente viables únicamente para producción a gran escala.

Los microcontroladores son circuitos integrados diseñados para ejecutar programas almacenados en memoria. Incluye CPU, RAM, ROM, puertos de entrada/salida, temporizadores, etc. A diferencia de los microprocesadores, que son dispositivos de propósito general, los microcontroladores están más enfocados a realizar un tipo de tarea

concreto para la cual están más optimizados. Suelen ser de baja potencia y bajo coste. Existe una amplia variedad de microcontroladores.

La estructura interna de las FPGAs las convierte en dispositivos perfectamente adecuados para realizar en paralelo tareas elementales de procesamiento digital. Además, su flexibilidad permite implementar no sólo coprocesadores de propósito específico, sino también interfaces, controladores, lógica de interconexión e incluso microprocesadores, posibilitando el co–diseño hardware/software eficiente. Simultaneamente al incremento de la riqueza y potencia de los recursos físicos disponibles ha ido la evolución de las herramientas de síntesis, de simulación y de diseño a nivel de sistema que participan en las diferentes etapas del flujo de diseño. Por todo ello, las FPGAs son hoy en día la opción más atractiva para el desarrollo de aplicaciones de procesamiento digital de señal.

Para elegir una FPGA frente a otros circuitos integrados hay que tener en cuenta sus ventajas respecto a estos:

FPGA vs CPLD:

- Una FPGA presenta una mayor densidad de lógica que los CPLDs.
- La FPGA presenta un menor precio por unidad de lógica, aunque el precio por circuito integrado sea mayor.
- Debido a que una FPGA presenta una mayor cantidad de lógica, las FPGAs permiten diseñar sistemas mucho más complejos que un CPLD.
- Las FPGAs llevan embebidos circuitos optimizados para realizar operaciones aritméticas, tales como contadores, sumadores, multiplicadores, bloques DSP, etc.

FPGA vs microcontrolador:

- Una FPGA puede realizar más operaciones por ciclo de reloj que un microcontrolador.
- Una FPGA puede ejecutar operaciones de forma paralela, aumentando de esta forma la velocidad de procesamiento de datos.

- Una FPGA puede ser reprogramada para cambiar completamente su funcionalidad, mientras que un microcontrolador ya tiene una circuitería y un conjunto de instrucciones estático.

Los lenguajes de descripción de hardware más empleados en el diseño de FPGAS son VHDL y Verilog. Ambos son lenguajes que permiten diseñar la FPGA desde un punto de vista abstracto, funcional, aunque también se puede definir la estructura del hardware a bajo nivel. Existen además componentes predefinidos, los IPs, descritos en estos lenguajes para simplificar el diseño de la FPGA. Los principales fabricantes de FPGAs proveen de herramientas para hacer más sencillo el proceso de diseño de una FPGA. Así, Xilinx Inc. ofrece la macroherramienta Xilinx ISE Design Suite, que consiste en un conjunto de herramientas destinadas al diseño de FPGAs (entre otros dispositivos electrónicos).

1.2. Objetivos de la tesis

Los objetivos que se plantean para esta tesis doctoral se describen a continuación. En primer lugar se estudiarán las diferentes arquitecturas existentes sobre la operación de convolución por ser la clave de muchas de las aplicaciones del procesamiento digital de señales. Esta operación está basada en la operación MAC (multiplica y acumula), por lo que se consultarán en la bibliografía disponible las distintas aportaciones para aumentar el rendimiento y la velocidad para esta operación. Además se analizarán los sumadores y multiplicadores disponible en la bibliografía.

Posteriormente se estudiarán las distintas posibilidades que se proponen con el hardware comercial (FPGA, DSP, GPP, etc.) para esta operación, aunque los estudios de esta tesis estarán basados en las FPGAs por ser las más eficientes.

A continuación se obtendrá una implementación particular de la convolución en FPGAs de bajo coste, utilizando aritmética CSA (*carry save adder*) y buffers circulares, proponiendo una arquitectura novedosa sobre los bloques de lógica configurable de una familia de FPGA comerciales de Xilinx que optimizan al máximo los recursos disponibles en este tipo de dispositivos, tales como la Spartan–3 y la Spartan–6, y su velocidad.

En último lugar se cuantificará una comparación de los resultados obtenidos de la operación de convolución en distintos dispositivos, en FPGAs de bajo coste (Spartan–3 y

Spartan–6 de Xilinx), en procesadores digitales de señal (DSP TMS320C6713 de Texas Intrument) y procesadores de propósito general en donde se toman como ejemplo dos procesadores ARM (ARM7 y ARM Cortex M3). El objetivo es validar las soluciones obtenidas sobre entornos de desarrollo comerciales y de tabular los resultados obtenidos.

1.3. Grado de innovación previsto

Las aplicaciones del procesamiento digital de señales se basan en las operación MAC (multiplica y acumula). Actualmente estas operaciones se realizan sobre procesadores específicos que disponen de un hardware dedicado a esta operación (procesadores digitales de señal). Una de las operaciones más utilizadas en este tratamiento es la convolución, que en las investigaciones actuales se está implementando en FPGAs donde, debido al rápido avance en la fabricación de dispositivos electrónicos, la solución adoptada en general es utilizar un dispositivo electrónico con mayor capacidad y velocidad.

Una aportación fundamental de esta tesis doctoral se basa en la propuesta de unas nuevas arquitecturas que optimicen los recursos hardware disponibles, de tal modo que los resultados obtenidos en FPGAs de bajo coste sean iguales e incluso mayores que los que se obtengan en FPGAs más potentes. Estos resultados serían extrapolables a su implementación en FPGAs más potentes.

La innovación fundamental de esta tesis doctoral es la aplicación de la aritmética CSA en FPGAs de bajo coste para el cálculo de la convolución, que hasta el momento no había sido utilizada. Se consiguen de esta manera, que en FPGAs con un coste económico y un consumo de potencia mucho menor, velocidades o rendimientos comparables a FPGAs más potentes. Esta optimización se realiza tanto en recursos consumidos como en velocidad.

1.4. Estructura de la memoria

Esta memoria de tesis está estructurada en ocho capítulos. En este primer capítulo se describe brevemente la historia del procesamiento digital de señales, desde sus comienzos hasta la fecha actual y sus aplicaciones principales,. A continuación se introduce la importancia de la operación de convolución en el procesamiento digital de señales. Posteriormente se realiza una descripción genérica de los dispositivos

electrónicos que más se suelen utilizar en este procesamiento y se justifica la utilización de las FPGA como dispositivo electrónico más eficiente, para terminar con los objetivos generales de la tesis y el grado de innovación previsto.

El capítulo 2 consiste en la descripción teórica de la operación de convolución, para lo cual se definen los sistemas lineales e invariantes en el tiempo (sistemas LTI), la función impulso y la función respuesta al impulso. Asimismo se define la convolución en tiempo continuo y su paso a tiempo discreto, y sus propiedades matemáticas más importantes. Termina este capítulo estudiando el cálculo de la convolución en tiempo discreto, describiendo los dos algoritmos más útiles para ello (desde el punto de vista de la señal de entrada y desde el punto de vista de la señal de salida).

El capítulo 3 se dedica al estudio de los sumadores y multiplicadores existentes en la bibliografía para justificar el empleo de la multiplicación en *carry-save* para el cálculo de la convolución, que constituye la innovación de esta tesis doctoral que es la propuesta de una arquitectura novedosa para la operación MAC (multiplica y acumula).

En el capítulo cuarto se realiza un estudio de los dispositivos electrónicos de bajo coste que se emplean actualmente para el procesamiento digital de señales, y en concreto en la convolución, como son los DSPs y las FPGAs. En primer lugar se introduce las características generales de los procesadores digitales de señal y a continuación se realiza un estudio más exhaustivo de las FPGA particularizando para las utilizadas en los datos experimentales que serán la Spartan–3 y la Spartan–6 de Xilinx, además este estudio se centra en los multiplicadores empotrados y módulos DSP que incluyen estos dispositivos.

En el quinto capítulo se describen las investigaciones realizadas sobre el cálculo de la convolución en FPGA, ya sean de la compañía que sean o de alto o bajo coste, se realiza un compendio sobre la arquitectura multiplica/acumula en FPGAs, los que estudian sobre sumadores y multiplicadores, en particular los estudios sobre el multiplicador *carry-save*. También se detallarán las FPGAs que contienen multiplicadores empotrados y bloques DSPs.

En el capítulo sexto se realiza la implementación de una arquitectura propuesta para operación de convolución en FPGAs de bajo coste, en concreto se realizarán estudios sobre la familia Spartan de Xilinx, que es la familia de bajo coste de la compañía mencionada. En concreto se realizarán los cálculos y resultados experimentales para la

obtención de la operación de convolución, cuya principal operación es multiplica-acumula, utilizando aritmética redundante, a partir de la multiplicación *carry–save*, que no ha sido utilizada en la bibliografía. Se Justificará experimentalmente el uso de esta aritmética cuando el número de operaciones MAC es elevado.

Para concluir con los datos experimentales en esta tesis, en el capítulo 7 se expone una comparación de los resultados obtenidos en la FPGA con otros dos tipos de dispositivos de bajo coste: con un procesador digital de señales (DSP) de Texas Instrument, en concreto el DSP TMS320C6713, que es un procesador de 32 bits de coma flotante, y dos microprocesadores empotrados de la familia ARM de 32 bits tales como son el ARM7 y el ARM Cortex M3.

En el capítulo 8 y último se expondrán las conclusiones de todos los estudios e investigaciones de esta tesis y se comentarán las principales aportaciones que se han realizado para la optimización de los recursos hardware para la operación de la convolución, dentro del campo del procesamiento digital de señales.

2 Operación de convolución

RESUMEN

Antes de definir la operación de la convolución, el actual capítulo comienza con la descripción de los sistemas lineales e invariantes en el tiempo, que son la mayoría que nos encontramos en aplicaciones de ingeniería. A continuación se define lo que en matemáticas se llama la función impulso (delta de Dirac) y la respuesta a ese impulso de un sistema lineal e invariante en el tiempo. Estos conceptos son necesarios para introducir matemáticamente la operación de la convolución en tiempo continuo y explicar el sentido físico que tiene la convolución en ingeniería, y su utilidad. Posteriormente se realiza el paso de la convolución en tiempo continuo a tiempo discreto, que será sobre la que se tiene que realizar el cálculo por los dos métodos expuestos en este capítulo: desde el punto de vista de la señal de salida y desde el punto de vista de la señal de entrada. Finalmente se enumeran algunas aplicaciones de la convolución en distintos ámbitos de la ciencia y de la ingeniería.

2.1. Sistemas lineales e invariantes en el tiempo

El estudio de los sistemas lineales e invariantes en el tiempo es de vital importancia para el tratamiento de la señal ya que básicamente cualquier sistema puede emularse bajo estas características; siendo su análisis de fácil comprensión y muy bien explorado. Esto no quiere decir que nunca nos toparemos con sistemas que no respondan a estas características, sin embargo la mayoría de los sistemas utilizados en diversas ramas de la ingeniería son de este tipo, como se indica en Proakis [4].

Se dice que **un sistema es lineal** si verifica el principio de superposición. Es decir, si $y_1[n]$ es la salida de un sistema ante una entrada $x_1[n]$, e $y_2[n]$ es la salida de un sistema ante una entrada $x_2[n]$, entonces, si el sistema es lineal, ante una entrada $x[n] = \alpha \bullet x_1[n] + \beta \bullet x_2[n]$, la salida es $y[n] = \alpha \bullet y_1[n] + \beta\, y_2[n]$ siendo α, β , $x_1[n]$ e $x_2[n]$ arbitrarios.

Lo anterior significa que, si la excitación se multiplica por una constante, la respuesta también se multiplicará por la misma constante. Además, si se le aplica la suma de dos excitaciones diferentes, la respuesta será la suma de las respuestas a cada una de las excitaciones aplicadas en forma independiente.

Un sistema que ante una entrada $x_1[n]$ produce una salida $y_1[n]$ es **invariante en el tiempo** si ante una entrada $x[n] = x_1[n–n_0]$ la salida es $y[n] = y_1[n−n_0]$. Es decir que si la secuencia de entrada se retarda o adelanta, la secuencia de salida estará retardada o adelantada ese mismo número de muestras. También podemos decir que la salida del sistema será la misma independientemente del instante en que se aplique la entrada.

Lo anterior implica físicamente que las características del sistema no cambian en el tiempo. Su respuesta cambia dependiendo de la excitación que se le aplique y la dinámica interna del sistema mismo, pero la forma en que responde a una excitación particular es siempre la misma, independiente del instante de tiempo en que se le aplique.

Existen varios métodos para describir la relación entre la entrada y la salida de los sistemas lineales e invariantes en el tiempo (sistemas LTI), cuando ambas se representan en función del tiempo. Un método es mediante ecuaciones diferenciales lineales (en tiempo continuo) o ecuaciones en diferencias de coeficientes constantes (en tiempo discreto). Otra forma de representar esta relación sería mediante un diagrama de bloques que representa al sistema como una interconexión de tres operaciones elementales: multiplicación, suma y desplazamiento en el tiempo para sistemas en tiempo discreto, o integración para sistemas en tiempo continuo. La tercera forma es la descripción mediante variables de estado, que corresponde a una serie de ecuaciones diferenciales o en diferencias de primer orden acopladas que representan el comportamiento del “estado” del sistema y una ecuación que relaciona el estado con la salida.

Pero el método más utilizado para representar esta relación es en términos de su respuesta al impulso. Para comprender que significa la respuesta al impulso es necesario definir primero lo que es un impulso.

2.2. Función impulso y respuesta al impulso

En ingeniería usualmente se maneja la idea de una acción ocurriendo en un determinado instante de tiempo. Puede ser una fuerza en ese punto o una señal en un instante temporal,

se hace necesario desarrollar alguna manera cuantitativa de definir este hecho. Esto nos lleva a la idea de un pulso unitario. Es probablemente la segunda señal más importante en el estudio de señales y sistemas después de la función exponencial compleja.

La función delta de Dirac es una abstracción matemática creada por el físico inglés Paul Dirac que se puede decir que no existe físicamente pero es de gran utilidad en las áreas de ingeniería y ciencias. Se aplica en muchas ramas de la ciencia en las cuales se describen procesos mediante modelización matemática. Esta función, conocida también como el impulso unitario o función delta es una función infinitamente estrecha, infinitamente alta, cuya integral tiene un valor unitario. La manera mas simple de visualizar este concepto es usar un pulso rectangular que va de $(a - \varepsilon/2)$ hasta $(a + \varepsilon/2)$ con una altura de $1/\varepsilon$. Al momento de tomar su límite cuando $\varepsilon \to 0$, podemos observar que su ancho tiende a ser cero y su altura tiende a infinito mientras que su área total permanece constante con un valor de uno. La función del impulso usualmente se escribe como **δ(t)**.

$$\int_{-\infty}^{\infty} \delta(t) \cdot dt = 1$$

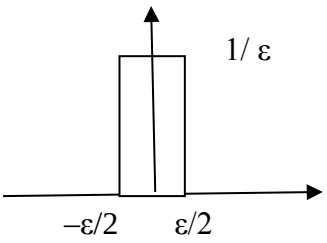


Figura 2-1 Función impulso.

Supongamos que un sistema lineal continuo e invariante en el tiempo no tiene energía almacenada internamente y se le aplica la excitación $x(t) = \delta(t)$, es decir, se la aplica como excitación un impulso unitario en tiempo cero con condiciones iniciales cero para el sistema. La respuesta del sistema para este caso se conoce como **respuesta al impulso** y se denominará **h(t)**.

A continuación se muestran varias excitaciones y sus correspondientes respuestas, cuando el sistema es lineal e invariante en el tiempo:

Tabla 2-1. Respuesta a un sistema LTI

Excitación	Respuesta
$\delta(t)$	$h(t)$
$a \cdot \delta(t)$	$a \cdot h(t)$
$\delta(t-t_0)$	$h(t-t_0)$

$a \cdot \delta(t-t_0)$	$a \cdot h(t-t_0)$
$a \cdot \delta(t-t_1) + b \cdot \delta(t-t_2)$	$a \cdot h(t-t_1) + b \cdot h(t-t_2)$

El objetivo ahora es obtener la respuesta del sistema a cualquier señal de entrada arbitraria x(t).

El primer paso para comprender los resultados que esta función nos ofrece, es examinar lo que sucede cuando esta función es multiplicada por alguna otra función.

$$f(t)\cdot\delta(t) = f(0)\cdot\delta(t)$$

Esta función es cero en todas partes excepto en el origen, así que básicamente estamos eliminando el valor de la función de multiplicación al evaluarla en cero. A primera vista esto no parece tener mayor importancia, porque ya sabemos que el impulso evaluado en cero es infinito, y todo lo multiplicado por infinito da un resultado infinito. Pero, si integramos el resultado de la multiplicación obtenemos el valor de la función en 0.

$$\int_{-\infty}^{\infty} f(t)\cdot\delta(t)\cdot dt = \int_{-\infty}^{\infty} f(0)\cdot\delta(t)\cdot dt = f(0)\cdot\int_{-\infty}^{\infty}\delta(t)\cdot dt = f(0)$$

Si hubiéramos usado $\delta(t - T)$ en vez de $\delta(t)$, podríamos haber desplazado $f(T)$. A esto es lo que llamaremos la propiedad de desplazamiento de la función de Dirac, el cual se usa normalmente para definir el impulso unitario. Esta propiedad es muy útil para desarrollar la idea de convolución la cual es una de los fundamentos principales para el procesamiento de señales. Al usar la convolución y esta propiedad podemos representar una aproximación de la salida de cualquier sistema si se conoce la respuesta al impulso del sistema y su señal de entrada.

La extensión de la función impulso unitario al tiempo discreto se convierte en una trivialidad. Todo lo que realmente necesitamos es darnos cuenta que la integración en tiempo continuo equivale a una sumatoria en tiempo discreto. Por lo tanto buscaremos una señal que al sumarla sea uno y al mismo tiempo sea cero en todas partes excepto en el origen.

La función impulso unitario, representada por $\delta(n)$, se define de la forma siguiente:

$$\delta(n) = \begin{cases} 1 & para\ n = 0 \\ 0 & para\ n \neq 0 \end{cases}$$

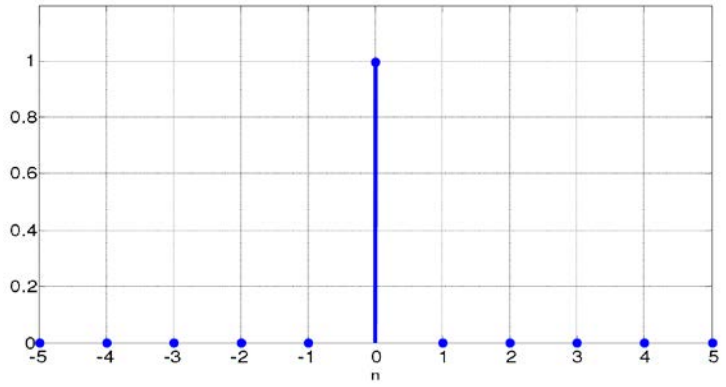


Figura 2-2 Impulso unitario discreto, δ(n).

La respuesta al impulso es exactamente lo que su nombre indica: es la respuesta de un sistema LTI, como por ejemplo un filtro, cuando la señal de entrada del sistema es un impulso unitario. Un sistema puede ser completamente descrito por su respuesta al impulso por las razones explicadas previamente, ya que todas las señales pueden ser representadas por una superposición de señales. Dada la respuesta al impulso h(n), determinamos la salida debido a una entrada arbitraria expresando la entrada como una superposición ponderada de impulsos desplazados en el tiempo. Por la linealidad e invarianza en el tiempo, las salidas deben ser una superposición ponderada de respuestas al impulso desplazadas en el tiempo. El término convolución se usa para describir el procedimiento en la determinación de la salida a partir de la entrada y la respuesta al impulso.

Una señal se define como una cantidad física que varía con el tiempo, el espacio o cualquier otra variable o variables independientes. Matemáticamente, una señal se expresa como una función de una o más variables independientes.

Los dos métodos más utilizados para el análisis de sistemas del comportamiento y la respuesta de un sistema lineal a una determinada señal de entrada son los siguientes. El primero de estos métodos se basa en obtener la solución de las ecuaciones en diferencias que relacionan la entrada con la salida del sistema. Generalmente esta ecuación tiene la forma:

$$y(n) = F[y(n-1), y(n-2), \ldots\ldots, y(n-N), x(n-1), \ldots\ldots\ldots\ldots, x(n-M)]$$

donde F[...] representa cualquier función de las cantidades entre corchetes. Por lo general los sistemas LTI responden a la ecuación de entrada – salida dada por:

$$y(n) = -\sum_{k=1}^{N} a_k \cdot y(n-k) + \sum_{k=1}^{M} b_k \cdot x(n-k)$$

donde a_k y b_k son parámetros constantes que especifican el sistema y son independientes de x(n) e y(n).

El otro método de análisis del comportamiento de un sistema lineal ante una determinada entrada se basa en descomponer dicha señal de entrada en señales elementales. Las señales elementales se escogen de manera tal que sea fácil determinar la respuesta del sistema a cada una de ellas. Utilizando la propiedad de linealidad del sistema, se suman las respuestas del sistema a la señal de entrada global.

La respuesta al impulso es la salida de un sistema LTI debido a una entrada de impulso aplicada en el tiempo t = 0 o n = 0. La respuesta al impulso caracteriza por completo el comportamiento de cualquier sistema LTI. Esto puede parecer sorprendente, pero es una propiedad básica de todos los sistemas LTI. La respuesta al impulso se determina a menudo a partir del conocimiento de la configuración y dinámica del sistema, o en el caso de un sistema desconocido, puede medirse aplicando un impulso aproximado en la entrada del sistema. La generación de una secuencia de impulsos en tiempo discreto para probar un sistema desconocido es directa. En el caso en tiempo continuo, un impulso real de ancho cero y amplitud infinita no puede generarse en realidad y suele aproximarse físicamente como un pulso de gran amplitud y ancho estrecho. Así la respuesta al impulso puede aproximarse como el comportamiento del sistema en respuesta a una entrada de elevada amplitud y extremada corta duración.

Si la entrada para un sistema lineal se expresa como una superposición ponderada de impulsos desplazados en el tiempo, entonces la salida es una superposición ponderada de la respuesta del sistema a cada impulso desplazado en el tiempo. Si el sistema es además invariante con el tiempo, entonces la respuesta del mismo a un impulso desplazado en el tiempo es una versión desplazada en el tiempo de la respuesta del sistema a un impulso. Consecuentemente la salida de un sistema LTI está dada por una superposición ponderada de respuestas al impulso desplazadas en el tiempo. Esta superposición ponderada recibe el nombre de sumatoria de convolución en sistemas en tiempo discreto y de integral de convolución en sistemas en tiempo continuo.

2.3. Convolución en tiempo continuo y sentido físico de la convolución

La convolución es una operación muy importante en el procesamiento digital de señales. Constituye una herramienta matemática poderosa que permite la obtención de resultados importantes que de otra manera serían muy difícil deducir.

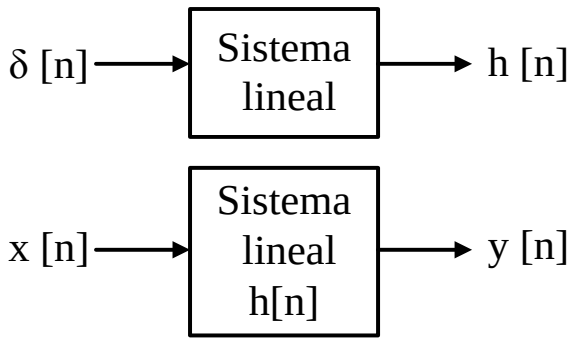
A continuación se describe el significado físico de la convolución como describe Smith [5]. Así como cualquier operación matemática, la convolución es un proceso que requiere dos cantidades; estas cantidades se conjugan en determinada forma y generan un resultado. La convolución es por tanto una operación matemática entre dos cantidades que, al realizarse, arroja una tercera cantidad. Cuando en alguna de sus formas, la presencia de una cantidad tiene influencia sobre la presencia de otra y viceversa, existe convolución de las dos cantidades. Es decir la interacción de dos cantidades en el tiempo y en el espacio es la convolución de esas dos cantidades. Esta interacción produce una tercera cantidad cuyas características están conformadas con características de ambas cantidades originales.

La convolución entre dos funciones es un concepto físico importante en muchas ramas de la ciencia. Sin embargo, como sucede con muchas relaciones matemáticas importantes, no es sencillo comprender sus alcances e implicaciones. Para el caso de sistemas lineales e invariantes en el tiempo, la integral de convolución permite determinar la respuesta del sistema ante cualquier entrada, a partir del conocimiento de la respuesta del sistema ante una única entrada particular, el impulso. Si la respuesta del sistema ante un impulso (la "respuesta impulsiva" del sistema) se nota como h(t), la salida de un sistema lineal e invariante en el tiempo (LTI) excitado con una entrada cualquiera x(t) está dada por la expresión:

$$y(t) = \int_{-\infty}^{\infty} x(\tau) \cdot h(t-\tau) \cdot d\tau = \int_{-\infty}^{\infty} x(t-\tau) \cdot h(\tau) \cdot d\tau$$

y se dice que la función y (t) es la convolución de las funciones x(t) y h(t) , que se representa por x (t) * h (t) . Esta expresión se conoce como la **integral de convolución**.

La convolución es una operación matemática formal que toma dos señales y produce una tercera señal. En sistemas lineales, la señal de salida y[n] del sistema se obtiene a partir de la convolución de la señal de entrada x[n] y de la respuesta h[n] a la función impulso δ[n] del sistema.



$$x[n] * h[n] = y[n]$$

Figura 2-3 Sentido físico de la convolución.

2.4. Paso de la convolución en tiempo continuo a tiempo discreto

Consideremos el caso de sistemas en tiempo discreto. En primer lugar expresamos una señal arbitraria como una superposición ponderada de impulsos desplazados en el tiempo. La sumatoria de convolución se obtiene entonces aplicando una señal representada de esta manera a un sistema LTI.

Consideremos el producto de la señal x[n] y la secuencia de impulsos δ[n], escrita de la forma siguiente:

$$x[n] \cdot \delta[n] = x[0] \cdot \delta[n]$$

Si generalizamos esta relación para el producto de x[n] y la secuencia de impulsos desplazados en el tiempo obtenemos lo siguiente:

$$x[n] \cdot \delta[n-k] = x[k] \cdot \delta[n-k]$$

En esta expresión n representa el índice de tiempo, por tanto x[n] denota una señal, en tanto que x[k] representa el valor de la señal x[n] en el instante k. Vemos que la multiplicación de una señal por el impulso desplazado en el tiempo produce un impulso desplazado en el tiempo con amplitud dada por el valor de la señal en el momento en que sucede el impulso.

Esta propiedad nos permite representar x[n] como una suma ponderada de impulsos desplazados en el tiempo.

$$x[n] = ...+ x[-2] \bullet \delta[n+2] + x[-1] \bullet \delta[n+1] + x[0] \bullet \delta[n] + x[1] \bullet \delta[n-1] + x[2] \bullet \delta[n-2]+...$$

Que de forma concisa podemos escribirla de la forma siguiente:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot \delta[n-k]$$

Si llamamos h[n] a la respuesta del sistema al impulso, la salida y[n] del sistema a una entrada x[n] es la convolución de ambas señales.

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Esta expresión es la llamada **sumatoria de convolución**. Esta fórmula nos indica que para el cálculo de la convolución necesitamos cuatro pasos: reflexión de h[k] para obtener h[–k], desplazamiento para obtener h[n–k], multiplicación y suma. La operación de reflexión se realiza una sola vez, en cambio las otras tres se repiten para todos los posibles valores del desplazamiento.

La suma de convolución provee una manera matemáticamente concisa para expresar el resultado de un sistema LTI, basado en una entrada arbitraria para una señal discreta y saber la respuesta del sistema.

2.5. Propiedades de la convolución

Las tres propiedades más importantes de la suma de convolución son las siguientes:

- Propiedad conmutativa:

La primera propiedad básica de la suma de convolución es que es conmutativa, es decir:

$$y[n] = x[n] * h[n] = h[n] * x[n]$$

Esto se demuestra en una forma directa mediante un cambio de variable m = n – k en la expresión de la sumatoria de convolución:

$$x[n]*h[n]=\sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]=\sum_{m=-\infty}^{\infty} x[n-m] \cdot h[m] =h[n]*x[n]$$

De acuerdo con esta última ecuación, la salida de un sistema LTI con entrada x[n] y respuesta al impulso h[n], es igual a la salida de un sistema LTI con entrada h[n] y respuesta al impulso x[n]. Este hecho se puede representar gráficamente así:

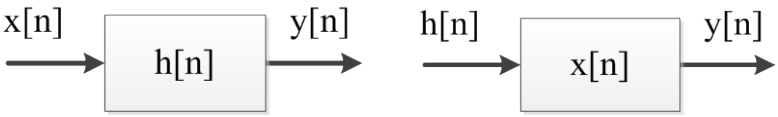


Figura 2-4 Representación de la propiedad conmutativa de la convolución.

- Propiedad asociativa:

Una segunda propiedad útil de la convolución es que es asociativa:

$$[x(n) * h_1(n)] * h_2(n) = x(n) * [h_1(n) * h_2(n)]$$

Para demostrar esta propiedad, sean $x[n] * h1[n] = f1[n]$ y $h1[n] * h2[n] = f2[n]$. Entonces:

$$f_1[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h_1[n-k]$$

y

$$\{x[n]*h_1[n]\}*h_2[n] = \sum_{m=-\infty}^{\infty} f_1[m] \cdot h_2[n-m] =$$

$$\sum_{m=-\infty}^{\infty} \left[\sum_{k=-\infty}^{\infty} x[k] \cdot h_1[m-k]\right] \cdot h_2[n-m]$$

Sustituyendo $r = m - k$ e intercambiando el orden de las sumatorias, tenemos:

$$\{x[n]*h_1[n]\}*h_2[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot \left[\sum_{r=-\infty}^{\infty} h_1[r] \cdot h_2[n-k-r]\right]$$

Y ahora, puesto que

$$f_2[n] = \sum_{r=-\infty}^{\infty} h_1[r] \cdot h_2[n-r]$$

tenemos

$$f_2[n-k] = \sum_{k=-\infty}^{\infty} h_1[r] \cdot h_2[n-k-r]$$

y, por lo tanto,

$$\{x[n]*h_1[n]\}*h_2[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot f_2[n-k] = x[n]*f_2[n] = x[n]*\{h_1[n]*h_2[n]\}$$

La interpretación de la propiedad asociativa se indica en la figura 2-4. Los sistemas mostrados en estos diagramas son sistemas LTI cuyas respuestas al impulso son las indicadas.

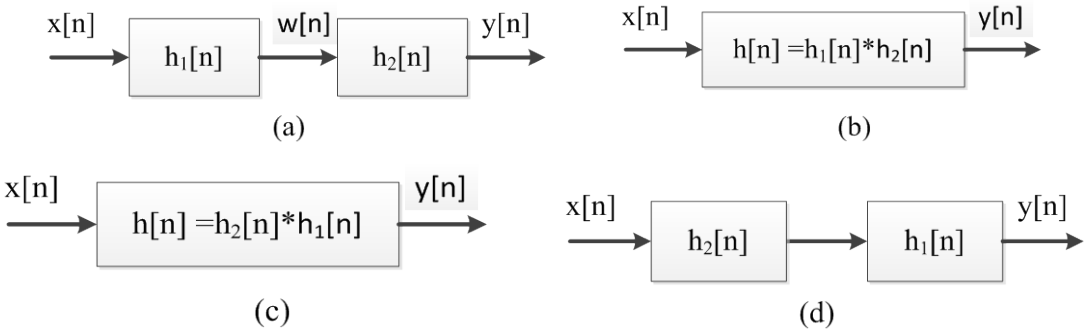


Figura 2-5 Representación de la propiedad asociativa de la convolución.

En la figura 2-4a,

$$y[n] = w[n] * h_2[n] = \{x[n]*h_1[n]\}*h_2[n]$$

En la figura 2-4b,

$$y[n] = x[n] * h[n] = x[n]*\{h_1[n]*h_2[n]\}$$

Según la propiedad asociativa, la interconexión en cascada de los dos sistemas en la figura 2-4a es equivalente al sistema único en la figura 2-4b. También, como consecuencia de la propiedad asociativa en conjunto con la propiedad conmutativa, la respuesta completa al escalón de sistemas LTI en cascada es independiente del orden en el cual los sistemas estén conectados (figuras 2-4 c y d).

- Propiedad distributiva:

Una tercera propiedad de la convolución es la distributiva con respecto a la suma, es decir,

$$x(n) * [h_1(n) + h_2(n)] = x(n) * h_1(n) + x(n) * h_2(n)$$

lo cual se verifica fácilmente usando la propiedad de linealidad de la suma.

Esta propiedad también tiene una interpretación útil. Consideremos los dos sistemas LTI conectados en paralelo de la figura 2-5a . Los dos sistemas $h_1[n]$ y $h_2[n]$ tienen la misma entrada y sus salidas se suman.

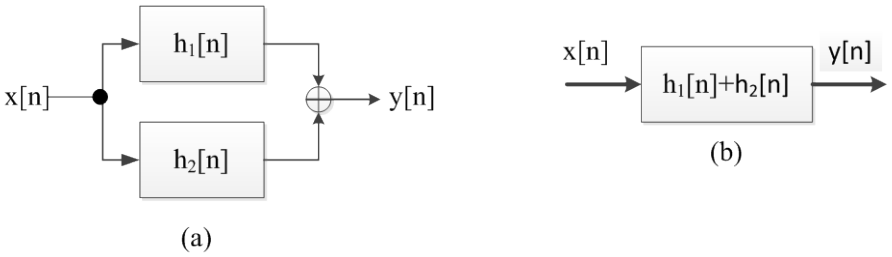


Figura 2-6 Representación de la propiedad distributiva de la convolución.

Como $y_1[n] = x[n]*h_1[n]$ e $y_2[n] = x[n]*h_2[n]$, la salida del sistema de la figura 2-6*a* es la siguiente:

$$y[n] = x[n] * h_1[n] + x[n] * h_2[n]$$

Y la salida del sistema de la figura 2-6b es

$$y[n] = x[n] * \{h_1[n] + h_2[n]\}$$

que corresponden a ambos miembros de la expresión de la propiedad distributiva. En consecuencia, por la propiedad distributiva de la convolución, una combinación en paralelo de sistemas LTI puede ser reemplazada por un sistema LTI cuya respuesta al impulso es la suma de las respuestas al impulso individuales de la combinación en paralelo.

Las tres propiedades pueden generalizarse para más de dos subsistemas.

2.6. Cálculo matemático de la convolución en tiempo discreto

En la mayoría de las aplicaciones del procesamiento digital de señales, la señal de entrada es del orden de cientos, miles o pocos millones de muestras en longitud, en cambio, la respuesta al impulso es mucho más corta, del orden de pocas muestras a cientos. La operación matemática de la convolución, no restringe la longitud de estas señales, sin embargo determina la longitud de la señal de salida, que es la suma de la longitud de las dos señales de entrada menos uno.

El cálculo de la convolución entre x(k) y h(k) supone la realización de los siguientes pasos:

- Reflexión. Se refleja h(k) respecto de k=0 para producir h(–k).
- Desplazamiento. Se desplaza h(–k), n hacia la derecha (izquierda) si n es positivo (negativo), para obtener h(n–k).
- Multiplicación. Multiplicamos x(k) por h(n–k) para obtener la secuencia producto.
- Suma. Se suman todos los valores de la secuencia, es decir, todas las respuestas impulsionales del sistema para obtener la respuesta en el punto indicado.

En el procesamiento digital de señales, la convolución puede ser entendida de dos formas distintas: desde el punto de vista de la señal de entrada y desde el punto de vista de la señal de salida [5]. Desde el punto de vista de la señal de entrada implica analizar cómo cada muestra de la señal de entrada contribuye en muchos valores de la señal de salida, proporciona una comprensión conceptual de cómo la convolución influye en el procesamiento digital de señales. Desde el punto de vista de la señal de salida evalúa cómo cada muestra en la señal de salida recibe información de muchas muestras de la señal de entrada.

Supongamos por ejemplo la convolución de dos señales x(n) y h(n) de longitudes M = 5 y N = 4.

$x(n) = [x_4, x_3, x_2, x_1, x_0]$ de longitud M = 5

$h[n] = [h_3, h_2, h_1, h_0]$ de longitud N =4

Por definición de convolución discreta:

$$y[i] = x[n] * h[n] = \sum_{j=0}^{M+N-1} x[j] \cdot h[i-j]$$

de longitud M + N – 1 = 5 + 4 – 1 = 8

Cada una de las muestras de la señal de salida, y(n) = {y7, y6, y5, y4, y3, y2, y1, y0}, se calcularía de la forma siguiente:

$y_0 = x_0 \cdot h_0$

$y_1 = x_0 \cdot h_1 + x_1 \cdot h_0$

$y_2 = x_0 \cdot h_2 + x_1 \cdot h_1 + x_2 \cdot h_0$

$y_3 = x_0 \cdot h_3 + x_1 \cdot h_2 + x_2 \cdot h_1 + x_3 \cdot h_0$

$y_4 = x_1 \cdot h_3 + x_2 \cdot h_2 + x_3 \cdot h_1 + x_4 \cdot h_0$

$y_5 = x_2 \cdot h_3 + x_3 \cdot h_2 + x_4 \cdot h_1$

$y_6 = x_3 \cdot h_3 + x_4 \cdot h_2$

$y_7 = x_4 \cdot h_3$

El primer punto de vista (*the input side algorithm*, [5]) realiza el cálculo de la convolución de las ecuaciones anteriores en vertical (por columnas). Analiza como cada muestra en la señal de entrada afecta a varias muestras de la señal de salida, es decir, toma x_0 y la multiplica por todas las muestras de la respuesta al impulso h_3, h_2, h_1, h_0. En el primer ciclo calcularía $y_0 = x_0 \cdot h_0$, a continuación va calculando los términos producto de cada muestra de salida: $x_0 \cdot h_1$, $x_0 \cdot h_2$, $x_0 \cdot h_3$, etc. Posteriormente toma x_1 y realiza el producto por todas las entradas h; es decir, calcula $x_1 \cdot h_0$, que al sumarlo con su primer término producto ya calculado obtiene la segunda muestra de la señal de salida y_1, calcula seguidamente $x_1 \cdot h_1$, $x_1 \cdot h_2$, $x_1 \cdot h_3$, etc. Continúa las operaciones con todas las muestras de la señal de entrada.

Este punto de vista de la convolución está basado en el concepto fundamental del procesamiento digital de señales: descompone la entrada, pasa los componentes por el sistema y sintetiza la salida.

El algoritmo que describe este cálculo es el siguiente:

```
For  i=0  to  M+N-1
    For  j=0  to  M
         y[i+j] =  y[i+j]  +  x[i] · h[j]
    End for
```

```
End for
```

El segundo punto de vista (*the output side algorithm*) realiza el cálculo de la convolución de las ecuaciones anteriores en horizontal (por filas). Examina como cada muestra de la señal de salida recibe información de muchos puntos de la señal de entrada. Este punto de vista describe la matemática de la operación de convolución. Puede calcularse el valor de cada muestra de la señal de salida independientemente del valor de las otras muestras.

El algoritmo utilizado para este cálculo lo indico a continuación:

```
For  i=0  to  M+N-1
    y[i] = 0
    For  j=0  to  N
    If (i-j ≥ 0)
    If (i-j < M)
          y[i] =  y[i]  +  x[j] · h[i-j]
    End for
End for
```

Estos dos algoritmos serán los utilizados en los cálculos y datos experimentales que se muestran en el capítulo 6.

2.7. Aplicaciones de la convolución

La convolución se encuentra inmersa en muchas de las aplicaciones que se utilizan en ingeniería, física y matemáticas. En la rama de las matemáticas, en estadística, un promedio móvil ponderado se representa mediante una convolución y en la teoría de la probabilidad, la distribución de probabilidad de la suma de dos variables aleatorias independientes es la convolución de cada una de sus distribuciones de probabilidad.

En física, allí donde haya un sistema lineal con un "principio de superposición", aparece una operación de convolución. En óptica, muchos tipos de desenfoques (o manchas) son descritos a través de convoluciones. Una sombra (e.j. la sombra en la mesa cuando tenemos la mano entre ésta y la fuente de luz) es la convolución de la forma de la fuente de luz que crea la sombra y del objeto cuya sombra se está proyectando. Una fotografía desenfocada es la convolución de la imagen correcta con el círculo borroso formado por el diafragma del iris (Jiménez et al. [6]). La convolución puede ser usada además para explicar por qué la difracción de una celosía de celdas unitarias en un espacio real da como resultado otra celosía de factores estructurados en un espacio recíproco. De

este modo si se calcula la transformada de Fourier de una serie de líneas paralelas, el resultado será una serie de puntos perpendiculares a las líneas y separados por una distancia inversamente proporcional al espacio entre dichas líneas (Read, [7]). En acústica, un eco es la convolución del sonido original con una función que represente los objetos variados que lo reflejan.

En química se aplica la convolución para calcular los factores de dispersión atómica, realizando esta operación entre la distribución de la densidad atómica con una función delta en la posición de un átomo.

En medicina se puede utilizar la convolución para reducir el ruido producido por la digitalización de las señales que resultan de un electrocardiograma (Mora et al. [8]). En este artículo se propone la implementación de diversas ventanas de convolución para obtener un filtro paso-baja utilizando funciones con una estructura matemática de tipo gaussiana, cuadrática, triangular y trigonométrica.

En economía también podemos encontrarnos ejemplos que aplican la convolución. Por ejemplo, Sadi Bin en [9] utiliza la convolución para la creación de un modelo estocástico que evalúe el capital económico para la concentración de riesgo crediticio, obteniendo una curva de distribución de pérdidas que afirma el nivel de pérdidas inesperadas de un portafolio.

En ingeniería eléctrica y otras disciplinas, la salida de un sistema lineal (estacionario o bien tiempo-invariante o espacio-invariante) es la convolución de la entrada con la respuesta del sistema a un impulso. Este será el sentido de la convolución que se tiene en cuenta en esta tesis.

3 Aritmética digital: sumadores, multiplicadores, operación MAC

RESUMEN

Una vez definida la operación de convolución en el capítulo actual se estudian los sumadores y multiplicadores disponibles en la bibliografía, ya que las operaciones de adición y multiplicación (operación MAC) son la base de esta operación. Comenzando con los sumadores más sencillos en los que ha de propagarse el acarreo se van describiendo las técnicas que existen para optimizar esta propagación, finalizando con el sumador carry-save, que obtendrá mejor rendimiento conforme vaya aumentando el número de sumandos y el número de bits de cada sumando. Con respecto a los multiplicadores, de forma análoga se comienza con los más sencillos, describiendo el procesador elemental que utilizan cada uno de ellos, hasta terminar con el multiplicador carry–save.

3.1 Cálculo de la convolución con sumadores y multiplicadores

Según se ha descrito en el capítulo anterior, para calcular la operación de convolución, necesitamos reflejar una de las señales, ir desplazando una de ellas, y realizar la operación MAC (multiplica-acumula). La operación de reflexión ha de realizarse solamente una vez, mientras que las otras tres han de realizarse la suma del número de muestras de cada una de las señales. El desplazamiento de las muestras de una de las señales puede realizarse en los dispositivos electrónicos mediante *buffers* circulares con los valores almacenados en memoria, pero este tiempo es significativamente menor que el empleado en la operación MAC. Por tanto para implementar la operación de convolución, las operaciones básicas que debemos optimizar son la multiplicación y la suma.

El multiplicador que se utiliza en esta tesis es el multiplicador *carry–save*, que aunque ha sido rechazado en los últimos años, se demostrará en esta tesis que cuando se tiene que

realizar gran cantidad de operaciones de multiplicación y adición, es útil, ya que el acarreo sólo ha de tenerse en cuenta en la operación final.

3.2 Sumadores binarios

Una de las muchas diferencias existentes entre la aritmética que utilizamos los humanos y la utilizada por los computadores es el número de cifras que se pueden utilizar. Los computadores solo pueden utilizar un número limitado de dígitos binarios. Debido a esto cuando se realizan operaciones con números reales en los computadores aparecen problemas de representación, precisión y redondeo. Además hay que tener en cuenta casos especiales como los números indeterminados y el infinito. El computador utiliza como base para realizar todos sus cálculos a la aritmética con números enteros, a partir de la cual realiza todas las operaciones ya sean para números reales como para enteros.

Debemos partir de la idea de que la manera de realizar las operaciones aritméticas los computadores es distinta a la que razonamos los seres humanos, ya que los computadores permiten construir algoritmos en los que los cálculos se realizan en paralelo, en contra del carácter secuencial que tiene la mente humana. Esta es la gran ventaja de los computadores respecto al cerebro humano, en la rapidez de realizar cálculos en base algorítmica.

Dentro de la aritmética usada por los computadores las más básicas son la suma y la resta, no solo por ser las más usuales, sino porque se utilizan en otras operaciones tales como el cálculo de direcciones de memoria, implementación de contadores, etc. Debido a esto es muy es conveniente realizarlas del modo más eficiente posible, tanto en rapidez como en recursos del computador utilizados.

A continuación se describen los cinco sumadores más utilizados en la actualidad. Para un estudio más detallado y otros tipos de sumadores puede consultarse el libro Ercegovac & Lang [10].

3.2.1. Sumador con propagación de acarreo (Carry Propagate Adder, CPA)

Partimos del circuito más sencillo consistente en un semisumador. Es un circuito aritmético que genera la suma de dos dígitos binarios. Debe tener dos entradas que son los sumandos (A, B) y dos salidas: una para la suma (S) y otra para el acarreo (C).

La tabla de verdad y las funciones algebraicas que debe cumplir este semisumador son las siguientes:

Tabla 3-1. Semisumador

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = A \cdot \overline{B} + \overline{A} \cdot B = A \oplus B$$

$$C = A \cdot B$$

El diagrama lógico para estas funciones sería el mostrado en la figura siguiente. También se suele representar como bloque indicando solo las entradas y salidas.

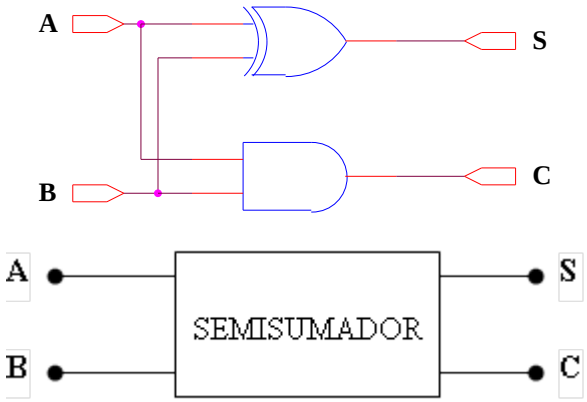


Figura 3-1 Semisumador

El sumador completo es un circuito combinacional que forma la suma aritmética de tres bits de entrada, y tiene dos salidas. Dos de las variables de entrada son los bits a sumar (A y B), y la tercera entrada Ci–1 es el acarreo de la posición menos significativa anterior. Las dos variables de salida son el resultado de la suma (S) y el acarreo (Ci) que se propagará a la etapa siguiente.

Las dos salidas las obtenemos de su tabla de verdad que son las siguientes:

Tabla 3-2. Sumador completo

A	B	C_{i-1}	S	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0

1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Para la suma se puede comprobar que la salida es 1 cuando corresponde a un número impar de unos, y puede demostrarse que equivale a la XOR de las tres variables de entrada.

$$S = \overline{A}\cdot\overline{B}\cdot C_{i-1} + \overline{A}\cdot B\cdot\overline{C_{i-1}} + A\cdot\overline{B}\cdot\overline{C_{i-1}} + A\cdot B\cdot C_{i-1} = (\overline{A}\cdot B + A\cdot\overline{B})\cdot\overline{C_{i-1}} + (\overline{A}\cdot\overline{B} + A\cdot B)\cdot C_{i-1} =$$
$$= (A \oplus B)\cdot\overline{C_{i-1}} + (\overline{A \oplus B})\cdot C_{i-1} = A \oplus B \oplus C_{i-1}$$

Para el acarreo de salida podemos simplificarlo de forma que utilizamos la salida de una puerta XOR de la suma S, de la manera siguiente:

$$C_i = A\cdot B + A\cdot\overline{B}\cdot C_{i-1} + \overline{A}\cdot B\cdot C_{i-1} = A\cdot B + (A\cdot\overline{B} + \overline{A}\cdot B)C_{i-1} = A\cdot B + (A \oplus B)\cdot C_{i-1}$$

De esta manera el circuito sumador completo queda de la forma siguiente:

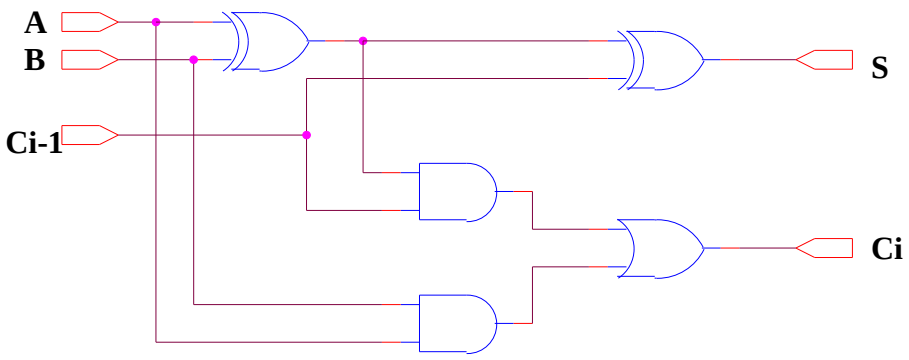
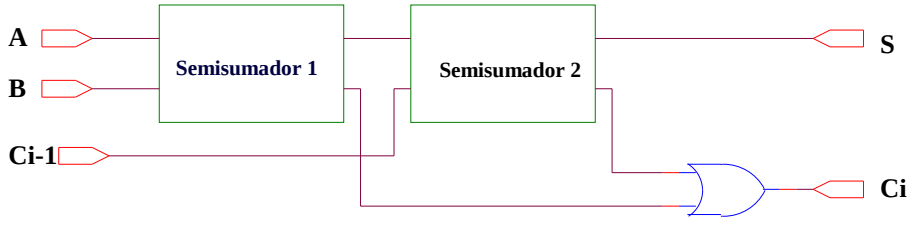


Figura 3-2 Sumador completo con puertas lógicas

También puede construirse a partir de dos semisumadores, para utilizar el circuito anterior en un diseño jerárquico, de la forma indicada en la figura 3-3:



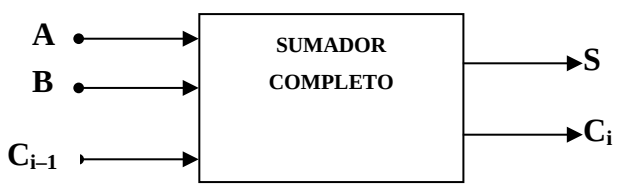


Figura 3-3 Sumador completo con semisumadores y símbolo lógico

El sumador paralelo con propagación de acarreo es un circuito combinacional que produce la suma aritmética de dos números binarios de n bits, utilizando n sumadores completos en paralelo, y se aplican simultáneamente (en paralelo) todos los bits para producir la suma.

Los sumadores completos se conectan de forma que la salida de acarreo del menos significativo está conectada a la entrada de acarreo del siguiente sumador completo.

Por ejemplo, para sumar dos números de 4 bits, tendremos el siguiente circuito:

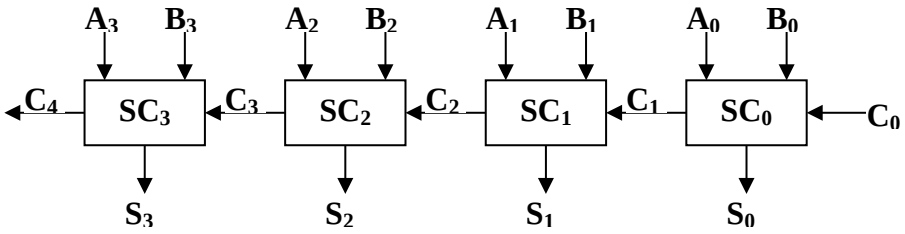


Figura 3-4 Sumador con propagación de acarreo (CPA) de 4 bits

Este circuito resultante de aplicar el método de "lápiz y papel" es el más sencillo pero también el menos eficiente desde el punto de vista del computador, ya que cada sumador completo tiene que esperar la salida de acarreo de la etapa anterior para realizar correctamente su suma.

Lo ideal sería pues, conocer los bits de acarreo sin necesidad de tener que esperar a que se realicen otras operaciones que no influyen en su obtención, con lo que se vería incrementada notablemente la velocidad de cálculo. Básicamente, acelerar la suma y la resta consiste en tratar de formas diferentes el acarreo de cada etapa, buscando métodos diferentes de proceso en paralelo para obtener los resultados con pocos recursos hardware y con el menor retardo posible. Esta es la diferencia fundamental en realizar el proceso secuencial (con lápiz y papel) o un proceso paralelo (computerizado).

Algunas de las técnicas para tratar el acarreo y obtener el resultado de la suma son: selección de acarreo (*carry select adder*), anticipación de acarreo (*carry look–ahead adder*), salto de acarreo (*carry skip adder*) y sumador con almacenamiento de acarreo

(*carry save adder*). Los tres primeros sumadores utilizan técnicas de propagación del acarreo y se basan en una propagación eficiente del mismo, en cambio el *carry save adder* (CSA) es un sumador sin propagación de acarreo que se usa cuando se quieren sumar más de dos operandos, tal y como requieren la operación de multiplicación, o cuando el ancho de palabra es mayor.

3.2.2. Sumador con selección de acarreo (Carry Select Adder)

Este sumador se basa en el cálculo en paralelo de los posibles resultados de la suma según se haya producido acarreo o no en la etapa anterior, una vez conocido el valor del acarreo de entrada correcto se toma el resultado apropiado, lo cual supone un tiempo de procesamiento menor que en el sumador de acarreo serie. Para implementar este sumador se dividen los bits de los operandos en bloques, para cada uno de estos bloques se calcula en paralelo e independientemente, el valor de la suma cuando el acarreo de entrada vale 0 y cuando vale 1, una vez conocido el acarreo correcto se selecciona mediante un multiplexor el resultado apropiado. De la misma manera se propaga el acarreo de salida al bloque siguiente, así sucesivamente hasta el último bloque. El acarreo de salida de la suma final es el seleccionado en el último bloque.

Para la elección de los bloques en los que se particiona la suma hay que tener en cuenta que cada uno de ellos puede estar implementado con cualquier tipo de sumador y que el número de bits de cada uno de los bloques puede ser variable. Estos dos aspectos son importantes para obtener una relación adecuada entre la velocidad del circuito resultante y el coste en hardware del circuito. Si queremos obtener mayor velocidad se debe tomar un tamaño tal que al multiplexar la señal de los bloques sobre los que se realiza la selección se estabilice lo más rápidamente posible, lo cual puede elevar el coste del circuito.

En la figura 3-5 presentamos un sumador de este tipo de 16 bits. La suma se realiza en paralelo en todos los módulos y a continuación se realiza la selección en serie. Con el bit C_4 se eligen los bits $S_{4\text{-}7}$ y C_8 reales, después con C_8 se eligen $S_{8\text{-}11}$ y C_{12}, y por último con C_{12} los bits $S_{12\text{-}15}$ y C_{16}.

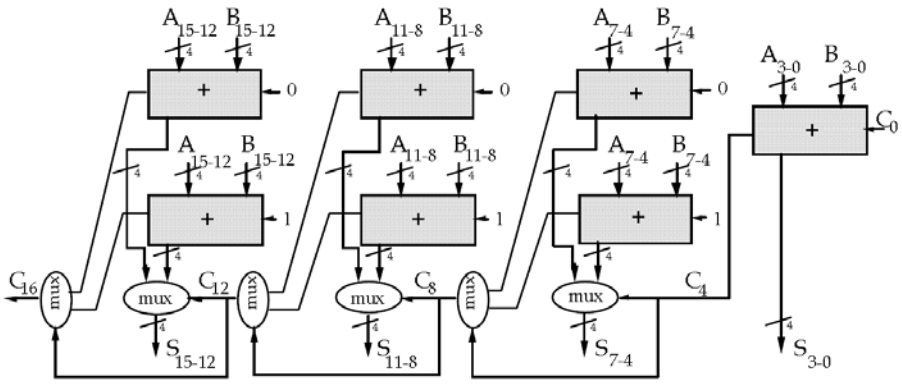


Figura 3-5 Sumador con selección de acarreo (*carry select adder*).

3.2.3. Sumador con anticipación de acarreo (Carry Lookahead Adder)

El sumador con anticipación de acarreo (*Carry Lookahead Adder*, CLA), se basa en la idea de que la suma es una función de tres parámetros (dos operandos de suma y un bit de acarreo inicial) donde se obtiene una solución invariante en el tiempo. Por esto podemos implementar la suma como una función combinacional en forma de suma de productos o de producto de sumas.

Para ver más claro este concepto recordamos como se genera y se propaga el acarreo. Supongamos la suma de dos bits A_i y B_i cuando el acarreo de entrada es C_i, el valor del acarreo de salida sería el mostrado en la tabla 3-3:

Tabla 3-3. Generación del acarreo

A_i	B_i	C_i	C_{i+1}	Función
0	0	0/1	0	No se genera
0	1	0/1	0/1	Si C_i es 1 se propaga el acarreo
1	0	0/1	0/1	Si C_i es 1 se propaga el acarreo
1	1	0/1	1	Se genera el acarreo

Basándonos en esta tabla definimos las funciones generador y propagador de acarreo:

$$G_i = A_i \cdot B_i$$

$$P_i = A_i \oplus B_i$$

$G_i = 1$ cuando $A_i = B_i = 1$ y se producirá un acarreo de salida independientemente de que exista o no acarreo en la posición anterior. Análogamente $P_i = 1$ cuando A_i o B_i sean igual a 1 pero no ambas, por lo que existirá acarreo de salida sólo si hay un acarreo de entrada; es decir, el acarreo de salida no se genera con la columna correspondiente, pero si lo hay en la entrada se propagará a través de la columna.

El valor de la suma podemos obtenerlo como sigue:

$$S_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i$$

Los acarreos que se producen en un módulo sumador de 4 bits con acarreo anticipado, pueden expresarse de la forma siguiente:

$C_{i+1} = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_i = G_i + P_i \cdot C_i$
$C_{i+2} = G_{i+1} + P_{i+1} \cdot C_{i+1}$
$C_{i+3} = G_{i+2} + P_{i+2} \cdot C_{i+2}$
$C_{i+4} = G_{i+3} + P_{i+3} \cdot C_{i+3}$

Sustituyendo los valores de los acarreos del sumador completo anterior, nos queda que los acarreos de cada uno de los sumadores pueden expresarse en función de los bits de entrada (sumandos) y del acarreo de entrada, sin necesidad de que el acarreo de cada sumador de un bit pase al siguiente; es decir anticipar donde se va a producir el acarreo.

$C_{i+1} = G_i + P_i \cdot C_i$
$C_{i+2} = G_{i+1} + P_{i+1} \cdot C_{i+1} = G_{i+1} + P_{i+1} \cdot (G_i + P_i \cdot C_i) = G_{i+1} + P_{i+1} \cdot G_i + P_{i+1} \cdot P_i \cdot C_i$
$C_{i+3} = G_{i+2} + P_{i+2} \cdot C_{i+2} = G_{i+2} + P_{i+2} \cdot G_{i+1} + P_{i+2} \cdot P_{i+1} \cdot G_i + P_{i+2} \cdot P_{i+1} \cdot P_i \cdot C_i$
$C_{i+4} = G_{i+3} + P_{i+3} \cdot C_{i+3} = G_{i+3} + P_{i+3} \cdot G_{i+2} + P_{i+3} \cdot P_{i+2} \cdot G_{i+1} + P_{i+3} \cdot P_{i+2} \cdot P_{i+1} \cdot G_i + P_{i+3} \cdot P_{i+2} \cdot P_{i+1} \cdot P_i \cdot C_i$
…….

Consecuentemente para generar el C_{i+1} no es necesario conocer el C_i, es suficiente conocer todos los P_i y G_i anteriores y el C_0. Dado que los bits P_i y G_i se pueden generar de forma independientemente para todos los bits todos los bits C_i se obtendrán a la vez. De esta forma desaparece la propagación de los bits de acarreo en serie.

Esta última ecuación, podríamos escribirla más reducida de la manera siguiente:

$$C_{i+4} = G_{(i,i+3)} + P_{(i,i+3)} \cdot C_i$$

donde:

$G_{(i,i+3)} = G_{i+3} + P_{i+3} \cdot G_{i+2} + P_{i+3} \cdot P_{i+2} \cdot G_{i+1} + P_{i+3} \cdot P_{i+2} \cdot P_{i+1} \cdot G_i$
$P_{(i,i+3)} = P_{i+3} \cdot P_{i+2} \cdot P_{i+1} \cdot P_i$

Como se observa ninguna de estas ecuaciones depende de los acarreos de las etapas anteriores: C_{i+1}, C_{i+2}, C_{i+3} o C_{i+4}. Este circuito constituye un generador de acarreo anticipado (CLA, *carry–look–ahead generator*). Este componente podría utilizarse para generar los acarreos adecuados en módulos mayores de 4 bits, por ejemplo para obtener un

sumador de 16 bits. Sin embargo, no es tan fácil debido a las limitaciones de conectividad de entrada y salida (*fan–in y fan–out*) de los circuitos integrados.

El circuito quedaría como se indica en la figura 3-6:

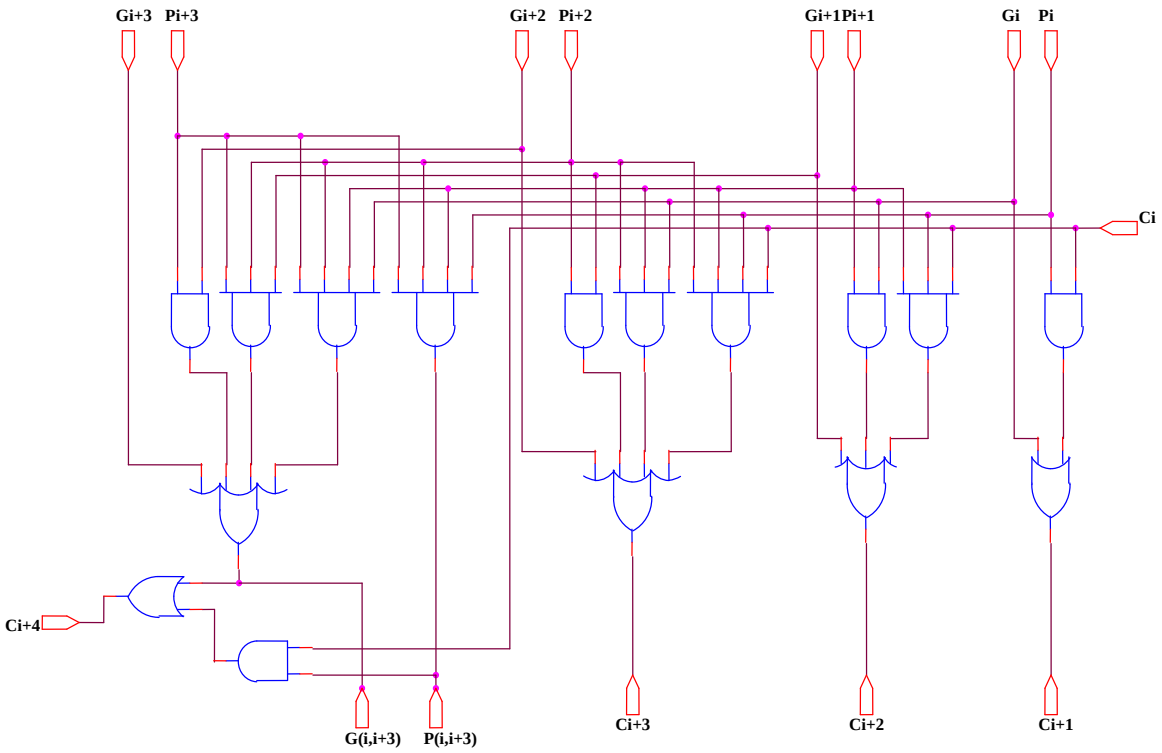


Figura 3-6 Circuito generador de acarreo anticipado (CLA, *carry–look–ahead generator*).

Utilizando el componente anterior como bloque funcional, un sumador binario de 4 bits con acarreo anticipado quedaría de la forma siguiente:

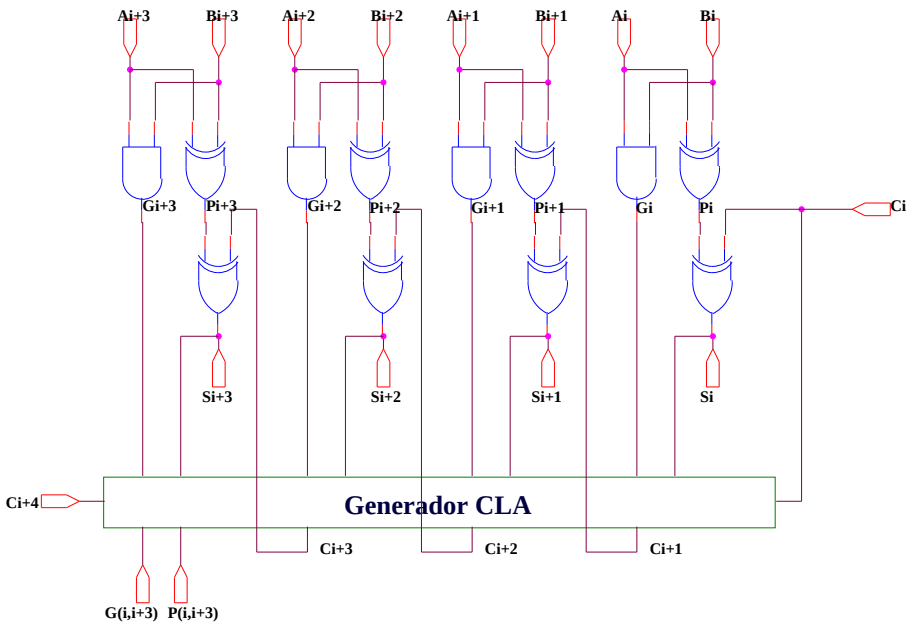


Figura 3-7 Sumador de 4 bit con anticipación de acarreo.

De esta manera se consigue la suma de manera muy eficiente, ya que el tiempo de respuesta es menor. Suponiendo que el tiempo de respuesta de una puerta AND y OR es Δ, tendremos los siguientes tiempos:

- Generar los P_i y G_i en paralelo, 2•Δ (función XOR)
- Generar, también en paralelo, todos los bits C_i , 2•Δ (función AND + OR)
- Generar la suma, todos los bits a la vez, 2•Δ (función XOR)

El tiempo de respuesta de un sumador CLA de 4 bits es 6•Δ, que además hemos conseguido que sea constante para cualquier número de bits. El análisis realizado presenta un problema en el tiempo de respuesta del circuito que genera los bits de acarreo. Aunque en teoría este tiempo es de 2•Δ (una puerta AND y una OR), en la práctica no es posible mantener ese tiempo cuando crece el número de bits, es decir, no tiene el mismo tiempo de respuesta una AND de 2 entradas que una de 8 entradas. Debido a esto es necesario limitar el tamaño del circuito a 2 ó 4 bits a lo sumo.

Una posibilidad para evitar este problema es combinar sumadores CLA con otros en serie, calculando algunos de los bits de acarreo en paralelo mientras que otros se calculan en serie. Esta es la idea del circuito sumador de 16 bits de la figura 3-8:

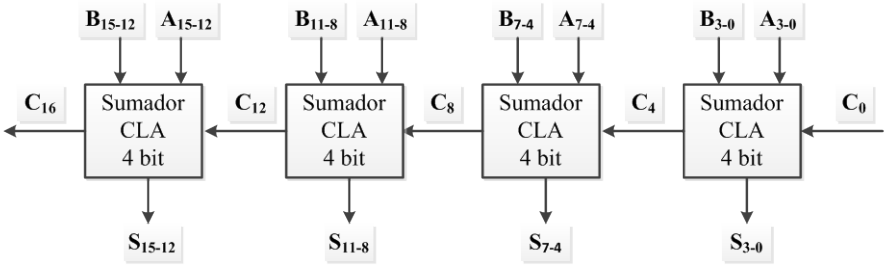


Figura 3-8 Sumador de 16 bit a partir de sumadores de 4 bits.

En este circuito los bits de acarreo internos se calculan en paralelo mediante un sumador CLA, mientras que los acarreos entre módulos (C_4, C_8, C_{12}) se transmiten en serie.

Otra posibilidad es organizar los circuitos CLA en forma de árbol, tal y como se muestra en la figura 3-9, donde aparece un sumador CLA de 8 bits, construido a partir de circuitos CLA de 2 bits.

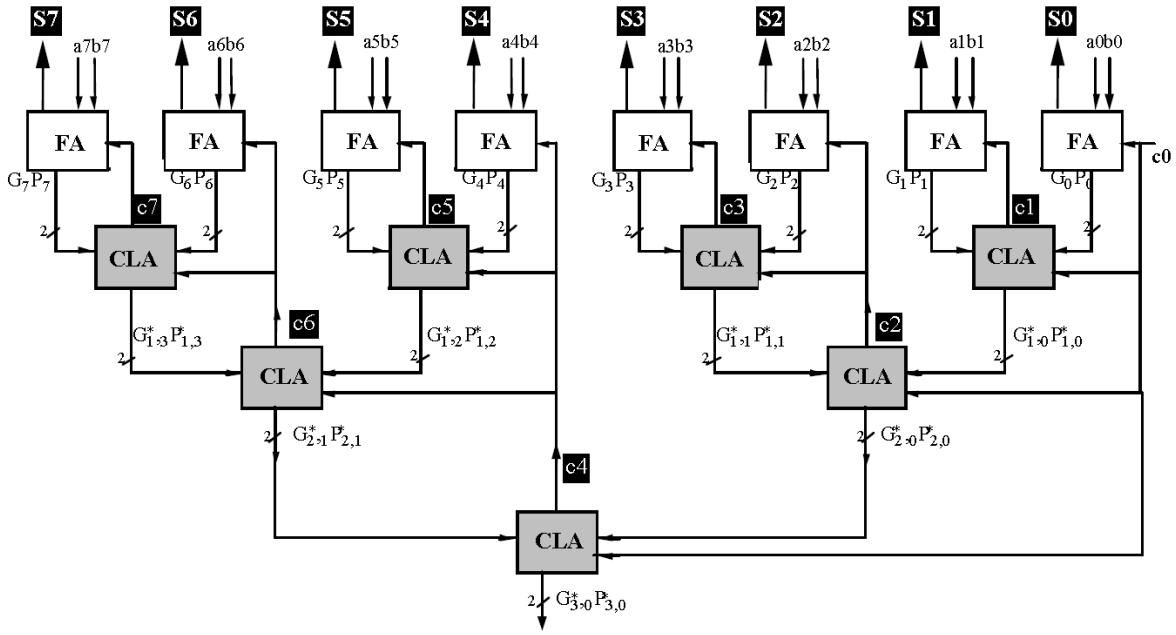


Figura 3-9 Sumador CLA de 8 bit a partir de sumadores CLA de 2 bits.

Evidentemente esta estructura se puede generalizar al número de bits que se desee, aunque los más utilizado son los circuitos CLA de 2 y 4 bits.

En resumen, los sumadores básicos tipo CPA son lentos pero baratos en cuanto a hardware utilizado, mientras que el sumador CLA es mucho más rápido pero también mucho más caro, debido a la cantidad de hardware que utiliza para generar los bits de acarreo.

3.2.4. Sumador con salto de acarreo (Carry Skip Adder)

Un sumador con salto de acarreo es, en parte, un sumador con propagación de acarreo y, en parte, un sumador con anticipación de acarreo. Es decir, si nos basamos en el hecho de que es mucho más simple calcular los bits P que calcular los bits G, podemos razonar como sigue. Si dividimos la suma en bloques, el acarreo de entrada de cada uno de ellos será la OR entre el bit del acarreo de salida y la AND lógica del valor P con el acarreo de entrada del bloque predecesor. De este modo, se puede saber el acarreo de entrada de cada bloque con cierta anticipación y hacer que cada uno de ellos trabaje en paralelo. Una vez el primer bloque genera el acarreo de salida, éste se propaga a los otros mediante una lógica sencilla. Si los operandos y los bloques son de n y k bits, respectivamente, y atravesar dos niveles lógicos se realiza en una unidad de tiempo, el bit de acarreo será el correcto en la entrada del i–ésimo bloque en k+(i–1) unidades de tiempo. Esto se debe a que el bit de acarreo generado por cada bloque se sabe en k unidades; mientras que el bit de acarreo que

proviene de la propagación de los bloques anteriores se conoce en (i–1) unidades de tiempo.

En la figura 3-10 se presenta un sumador *carry–skip* de 16 bits formado por sumadores serie de 4 bits.

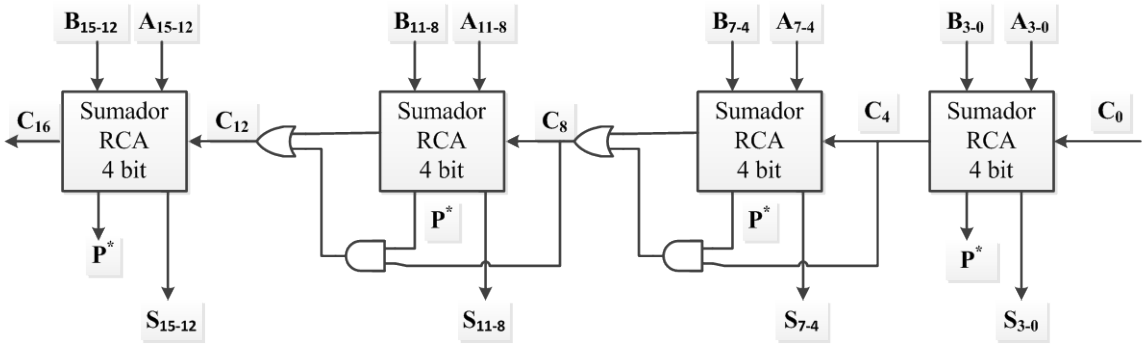


Figura 3-10 Sumador con salto de acarreo de 16 bits.

Este tipo de sumadores se suelen emplear en circuitos donde la tecnología permita la propagación de señales de una forma rápida. Por otro lado, como sucedía en un sumador con selección de acarreo, si se quiere optimizar el proceso, se puede coger un tamaño por bloque variable. Por regla general, si en un sumador con salto de acarreo se hacen los bloques internos mayores, se consigue acelerar el sumador.

3.2.5. Sumador con almacenamiento de acarreo (Carry Save Adder)

La representación redundante de números se utiliza para reducir el tiempo de suma, limitando el camino de la cadena de acarreo a varios bits. De esta manera el tiempo empleado para la realización de la suma no depende del número de bits de los operandos. Las representaciones más habituales son *carry–save* (CSA) y *signed-digit* (SD). La aritmética *carry–save* se utiliza ampliamente cuando se requieren sumar un número de operandos elevado.

Todos los sumadores vistos anteriormente, excepto el CPA, están diseñados para poder acelerar el proceso de acarreo y aliviar el retardo en la obtención del resultado de la suma. Es decir, su diseño les permitía trabajar en paralelo internamente. El sumador con almacenamiento de acarreo o Carry Save Adder (CSA) sigue una filosofía distinta. No persigue conseguir realizar sumas utilizando operaciones internas paralelas, sino ser la unidad de procesamiento en paralelo para circuitos aritméticos más complejos. En esta línea, no se puede considerar que el CSA sea un circuito sumador por sí mismo, ya que es

incapaz de devolver una palabra de suma y un bit de acarreo de salida como respuesta a la suma de dos operandos y un acarreo de entrada.

Por tanto, ni es un sumador completo ni es un semisumador. El *Carry Save Adder* posee tres palabras de entrada y dos palabras de salida. Una de las respuestas es la suma de cada uno de los bits i–ésimos de los tres operandos por separado. Mientras que la otra está compuesta por cada uno de los bits de acarreo generados en esas sumas. Para obtener el resultado completo se ha de combinar la palabra de suma parcial con la palabra de acarreo asociada a dicha suma, propagando los acarreos producidos. Esta última operación se lleva a cabo en un sumador con propagación de acarreo. Como se dijo anteriormente, este tipo de sumador es utilizado en casos especiales. Se hace uso del mismo cuando se quieren sumar más de dos operandos (método muy usado en circuitos multiplicadores), ya que permiten postergar la operación de acarreo hasta el final. Así pues, su utilidad no se verá con claridad hasta explicar la aceleración de la multiplicación.

El objetivo de un sumador CSA es acelerar la suma cuando se tienen que sumar más de dos operandos, tratando de evitar la propagación del acarreo. En este sumador, al sumar dos operandos, los acarreos no se propagan, sino que se usan como tercer operando en la suma siguiente. Sólo en la última suma habrá que propagar los acarreos.

En el caso de la convolución de dos señales, hay que realizar gran cantidad de sumas, por lo que la utilización de este tipo de sumador quedará más que justificada, ya que el tiempo empleado en la suma final es mucho más pequeño que las sumas parciales.

Veamos un ejemplo en decimal para describir el funcionamiento de esta técnica con la suma de tres números.

Tabla 3-4 Ejemplo de suma en CSA

Con propagación de acarreo (CPA)	Con almacenamiento de acarreo (CSA)
357	357
+ 409	409
766	CSA + 143
+ 143	899 ← PseudoSuma (PS)
909	001 ←Acarreo Salvado (AS)
+ 112	CSA + 112
1021	911 (PS)
	Suma final + 11 (AS)
	1021

La unidad básica de un sumador con acarreo almacenado es un sumador completo de un bit, pero el acarreo no se transmite de un sumador a otro, sino que se trata la entrada de acarreo como una entrada más y se aprovecha para introducir un nuevo operando. Como resultado obtenemos dos vectores de n bits, la pseudosuma parcial (PS) y el acarreo salvado (AS). La etapa básica de la suma sería la indicada en la figura 3-11:

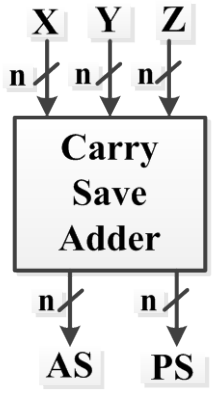


Figura 3-11 Sumador CSA de n bits.

El sumador CSA básico realiza la suma de tres operandos utilizando un array de sumadores completos de un bit, pero sin conectar la cadena de acarreo, como se muestra en la figura 3-12. El resultado es un número redundante en representación CS que está compuesta de una palabra de suma (PS) y una palabra de acarreo (AS).

Y el esquema del circuito, la figura 3-12, donde cada una de las celdas individuales del CSA son sumadores completos, en los que el acarreo se utiliza como tercer sumando.

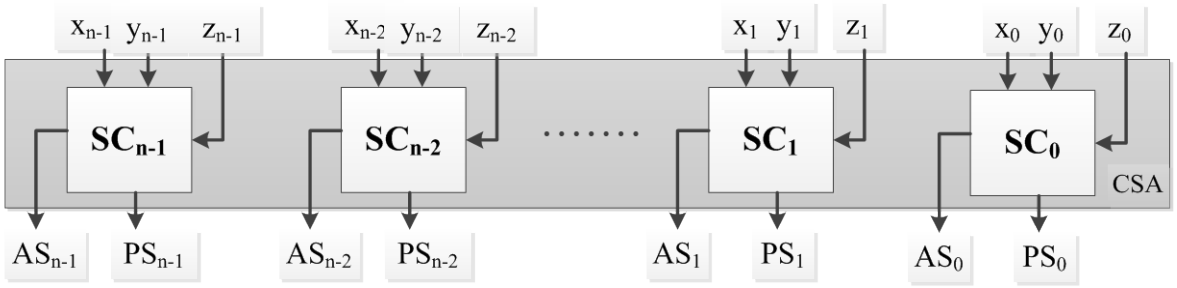


Figura 3-12 Esquema del circuito CSA de n bits.

Hay que tener en cuenta que de este modo no se obtiene la suma de tres operandos, dado que todavía no se han propagado los bits de acarreo. Se han sumado tres números y se han obtenido dos que a su vez hay que sumar, esta vez sí, propagando el acarreo en serie de un bit a otro. Por lo tanto hasta llegar a la suma final, los números se suman en un tiempo mínimo dado que no se propaga el acarreo entre los bits. Así pues, es posible sumar

muchos operandos eficientemente utilizando circuitos CSA, ya sea utilizando un único sumador CSA, o muchos organizados en estructuras tipo árbol.

Por ejemplo el circuito correspondiente a un sumador *carry–save* de 4 operandos de 4 bits sería el siguiente:

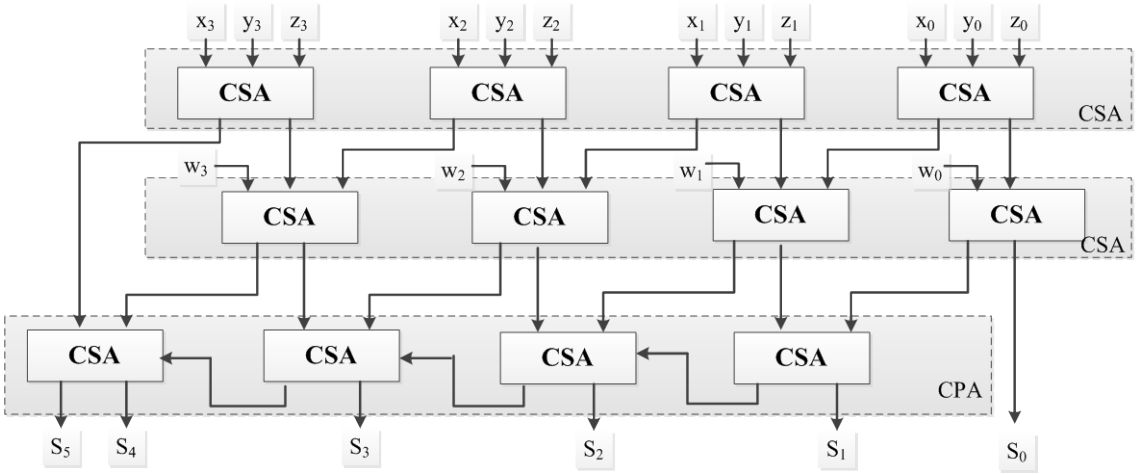


Figura 3-13 Sumador *carry–save* de 4 operandos de 4 bits.

Hay dos inconvenientes para los sumadores sin propagación de acarreo. Primero, requieren más hardware, porque debe haber una copia del registro, para que contenga las salidas de acarreo del sumador. Segundo, después del último paso, la palabra de orden superior del resultado se debe conectar a un sumador ordinario para combinar las partes de suma y acarreo. Pero la gran ventaja que tienen los sumadores CSA es que la velocidad de cálculo aumenta respecto a los CPA, conforme aumenta el número de sumandos y el número de bits de cada sumando.

Los números AS y PS de n–bits se obtienen sin propagación de acarreo con el retardo de un solo sumador completo (SC). Esta representación se le llama redundante, puesto que muchas combinaciones de AS y PS producen el mismo número. En la figura 3-14 se puede observar como al sumar dos números distintos que dan el mismo resultado en aritmética convencional producen dos combinaciones distintas de AS y PS. Por tanto hasta que no se realice la suma AS y PS no se tendrá un número en representación convencional. Para ello será necesario utilizar un sumador convencional que sume dos operandos y dé como resultado un único número. Mientras todas las operaciones se realicen en aritmética CSA no es necesaria realizar la conversión.

	1 0 1 0	10			0 1 0 1	5
+	0 0 1 0	+ 2		+	0 1 1 1	+ 7
PS	1 0 0 0	8		PS	0 0 1 0	2
AS	0 1 0 0	4		AS	1 0 1 0	10

Figura 3-14 Ejemplo de suma binaria en CSA.

Este circuito CSA desde otro punto de vista realiza la reducción de tres números binarios a dos números binarios, por lo que se le llama compresor [3:2] o contador [3:2]. Si se desea sumar dos números *carry–save* se necesita una reducción de 4 a 2. Esto se puede realizar utilizando dos compresores [3:2] como muestra la figura 3-15, llamado compresor [4:2]. En este caso el tiempo de cálculo para la suma de dos números de n–dígitos *carry–save* es el de dos sumadores completos. Obsérvese como la propagación de acarreo se corta y como el acarreo propagado del primer nivel se debe conectar a la entrada de acarreo de entrada del segundo.

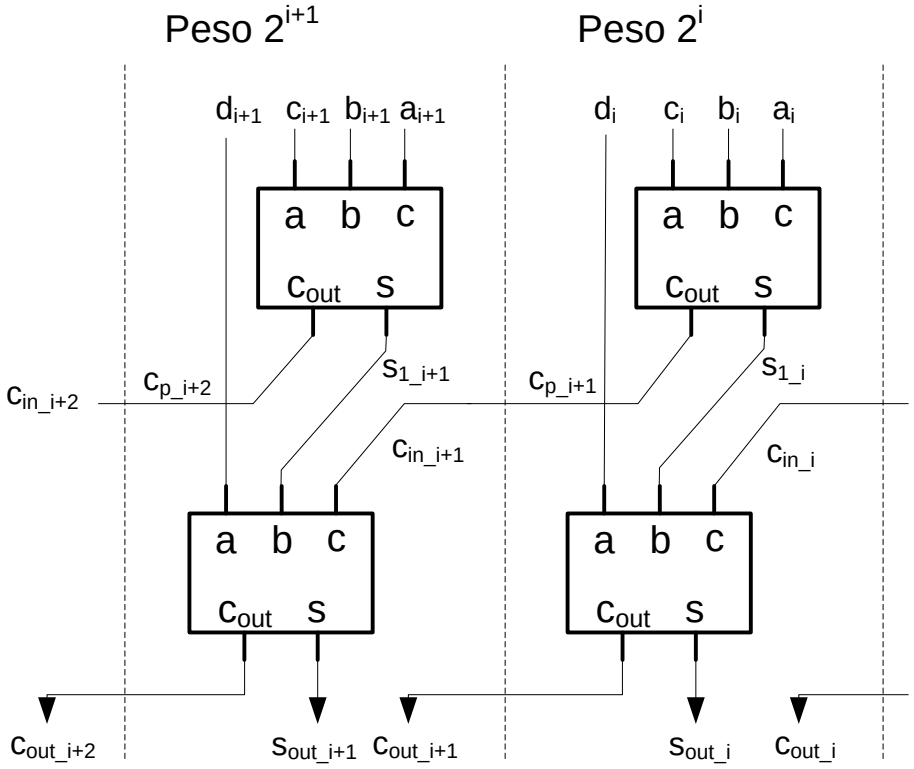


Figura 3-15 Compresor [4:2] construido a partir de dos compresores [3:2].

Se puede sumar un bit de m operandos y producir una palabra de suma PS y otra de acarreo AS. De este modo se tienen compresores [5:2] que realiza una reducción de cinco bits a dos, compresores [6:2] que realiza una reducción de 6 bits de entrada a dos y en

general compresores [p:t] que realiza la suma de p bit de entrada y produce una salida de t bits.

Una de las desventajas de la representación CS es que el número de bits involucrados en la suma es el doble. Para reducir el hardware una alternativa es la utilización de un radix superior, como proponen Beuchat y Muller [11], para optimización en FPGAs.

La utilización de la representación CS es especialmente útil en algoritmos que requieran muchas sumas intermedias puesto que todas las sumas se pueden llevar a cabo en representación CS y donde la conversión a representación convencional no consuma el tiempo ahorrado en la representación CS. Por ejemplo, en operaciones de convolución, acumulación, suma de multioperandos, multiplicación, división, raíz cuadrada, etc.

3.3 Multiplicadores binarios

La aparición y popularización de los sistemas de diseño con circuitos de lógica programable hacen posible en la actualidad que muchos algoritmos se realicen en hardware de forma más rápida y económica. Esto trae como consecuencia un interés renovado en la aritmética binaria, en particular en la multiplicación, por ser una de las operaciones más utilizadas en el procesamiento digital de señales.

Existe una gran cantidad de estudios donde se proponen distintas realizaciones de multiplicadores, aunque las aportaciones fundamentales en este tema tienen su origen en la década de los 60. Entonces los objetivos fundamentales eran la estandarización de componentes digitales y el desarrollo de unidades aritméticas potentes para ordenadores. La mayoría de estos artículos siguen teniendo validez que supera al marco tecnológico para el cual fueron pensados.

En el artículo "*A Suggestion for a Fast Multiplier*" [12], C. S. Wallace señala los tres aspectos en los que trabajar para hacer más rápida la multiplicación de dos números: acelerar la formación de productos parciales, reducir el número de productos parciales y acelerar la suma de productos parciales.

La primera de estas opciones conduce a un array de puertas simultáneamente. Uno de los primeros circuitos que implementa esta idea es el multiplicador a válvulas realizado por Jan Rajchman en 1940 [13].

La segunda de las opciones fue desarrollada por Booth en 1951 [14], aunque en la actualidad se usa una modificación de este algoritmo propuesta por Rubinfield [15], llamado normalmente algoritmo de Booth modificado (MBA, *Modified Booth Algorithm*). Este algoritmo consiste en recodificar uno de los operandos (el multiplicador) para reducir el número de productos parciales. Este operando se divide en grupos de 3, 4, o más bits, solapados entre sí en un bit y que se convierten, utilizando una tabla de verdad, a dígitos con signo que indican las transformaciones a realizar sobre el otro operando (el multiplicando). La suma de las sucesivas transformaciones sobre el segundo operando da el producto. El algoritmo de Booth mantiene su validez cuando los operandos están representados en complemento a 2. Aunque se han realizado estudios teóricos sobre esquemas que utilizan grupos de un mayor número de bits (Sam & Gupta, [16]), es preferible recodificar el operando dividiéndolo en grupos de 3 bits (Cooper, [17]) por dos razones. La primera de ellas es que los dígitos recodificados dependen solamente de un pequeño grupo de bits del operando que se recodifica (multiplicador). La segunda consiste en que las transformaciones a realizar sobre el otro operando (multiplicando) son simples (desplazar, complementar, etc.)

La idea de Booth presenta como contrapartida un aumento en la longitud de los productos parciales a sumar y una pérdida de regularidad.

La tercera de las sugerencias anteriores propone acelerar la suma de productos parciales, para lo cual el esquema más habitual es el Arbol de Wallace, formado por un conjunto de *full-adders* (también llamados sumadores *carry-save*, CSA o sumadores 3:2), donde el retardo de la suma es proporcional al logaritmo del número de sumandos. Un inconveniente de los árboles de Wallace es su poca regularidad problema que fue resuelto por Santoro & Horowitz, [18].

Por otro lado, Dadda [19] en 1965, propuso un esquema de multiplicación para acelerar esta suma. En este artículo propuso la idea de utilizar contadores paralelos con objeto de obtener la suma de los bits que forman las columnas de la matriz de productos parciales. Un contador paralelo es un circuito combinacional cuya salida es el número de “unos” que hay en su entrada. Por razones de simplicidad es aconsejable limitarse al uso de contadores [3:2] y no utilizar contadores paralelos de un gran número de entradas, aunque ello aumente el número de etapas necesarias como hacen Cappello & Steiglitz en [20], o Dhurkadas en [21] para conseguir esta reducción.

La diferencia existente entre los métodos utilizados por Wallace y Dadda es que Wallace se centra en las filas de la matriz de productos parciales y Dadda se fija en las columnas, de forma que a la solución de Wallace lo podemos llamar "aproximación horizontal" al problema de la multiplicación y al de Dadda "aproximación vertical".

Los multiplicadores se pueden clasificar según la forma de introducir los operandos según Boemo et al. en [22], o Teixeira et al. en [23]. Se denominan serie/serie si los dos operandos se introducen en serie, cuya ventaja es que utilizan un bajo número de entradas/salidas y que tienen un bajo consumo de recursos lógicos. Como desventaja es que están limitados a aplicaciones de baja velocidad tales como se describe en Nibouche et al. [24], o Aggoun et al. en [25] y [26]. Si sólo uno de los dos operandos ingresan en serie se denominan serie/paralelo, un ejemplo se encuentra en Bouridane et al. en [27]. Estos multiplicadores se utilizan en aplicaciones de velocidad media y presentan un consumo de recursos lógicos moderado. Por último si ambos operandos son introducidos de forma paralela, se llaman multiplicadores paralelo/paralelo.

Por otro lado, los multiplicadores se pueden clasificar según el algoritmo de cálculo en: suma y desplazamiento, por árbol y contadores. Los primeros utilizan el método clásico de “lápiz y papel” para calcular el producto., y normalmente se realiza construyendo una celda de procesamiento básica que se repite las veces necesarias. La comunicación entre estas celdas puede ser local (solamente existe comunicación entre celdas adyacentes) o global (existiendo conexión entre celdas alejadas). El algoritmo de árbol, nombrado anteriormente, fue propuesto por Wallace [12], donde el retardo de la suma es proporcional al logaritmo del número de sumandos. La tercera forma de acelerar los productos parciales se basa en la utilización de contadores paralelos para obtener la suma de los bits que forman las columnas de la matriz de los productos parciales, propuesta por Dadda [19].

Se llaman multiplicadores paralelos a los que realizan el algoritmo de suma y desplazamiento de forma simultánea, que realizan la multiplicación muy rápidamente, pero son difíciles de implementar cuando el dispositivo tiene recursos limitados. Más complicado resulta cuando los operandos están expresados en el formato de coma flotante, que comunmente utilizan este tipo de multiplicadores para el producto de las mantisas (Shirazi et al. [28], y Ligon et al. [29]). Conforme aumenta la complejidad de la operación, se incrementa el consumo de recursos lógicos utilizados y, por tanto, los costes de implementación.

Una forma de implementar los multiplicadores paralelos, es la multiplicación secuencial como indican Patterson [30] y Hayes [31] que utiliza un menor número de recursos lógicos a base de ejecutar tantas iteraciones como longitud de palabra tienen los operandos (Ordóñez et al. [32], Parhami [33]). El hecho de utilizar un excesivo tiempo de cálculo representa una seria desventaja respecto a otras estructuras, por lo que estos multiplicadores son poco utilizados en las aplicaciones típicas de procesamiento digital de la señal.

Existen numerosos esquemas de multiplicadores paralelos, y sus variaciones radican en la forma de abordar la ejecución de los productos parciales para obtener una mayor velocidad de procesamiento. Vamos a estudiar algunos de ellos:

3.3.1. Multiplicador Ripple Carry

El multiplicador paralelo más difundido está basado en un esquema de propagación de acarreo. Está basado en un array de sumadores de acarreo propagado y constituye una primera aproximación para implementar el algoritmo de sumas sucesivas y desplazamientos. Este multiplicador tiene la característica de transferir la propagación del acarreo a la siguiente suma parcial hasta que terminen los productos parciales correspondientes a esta fila, en este instante el acarreo generado se propaga al último producto de la siguiente fila de productos parciales y así sucesivamente hasta completar la multiplicación.

El bloque principal es un sumador completo de un bit. En la figura 3-16 se muestra un multiplicador *Ripple Carry* para dos operandos de 4 bits.

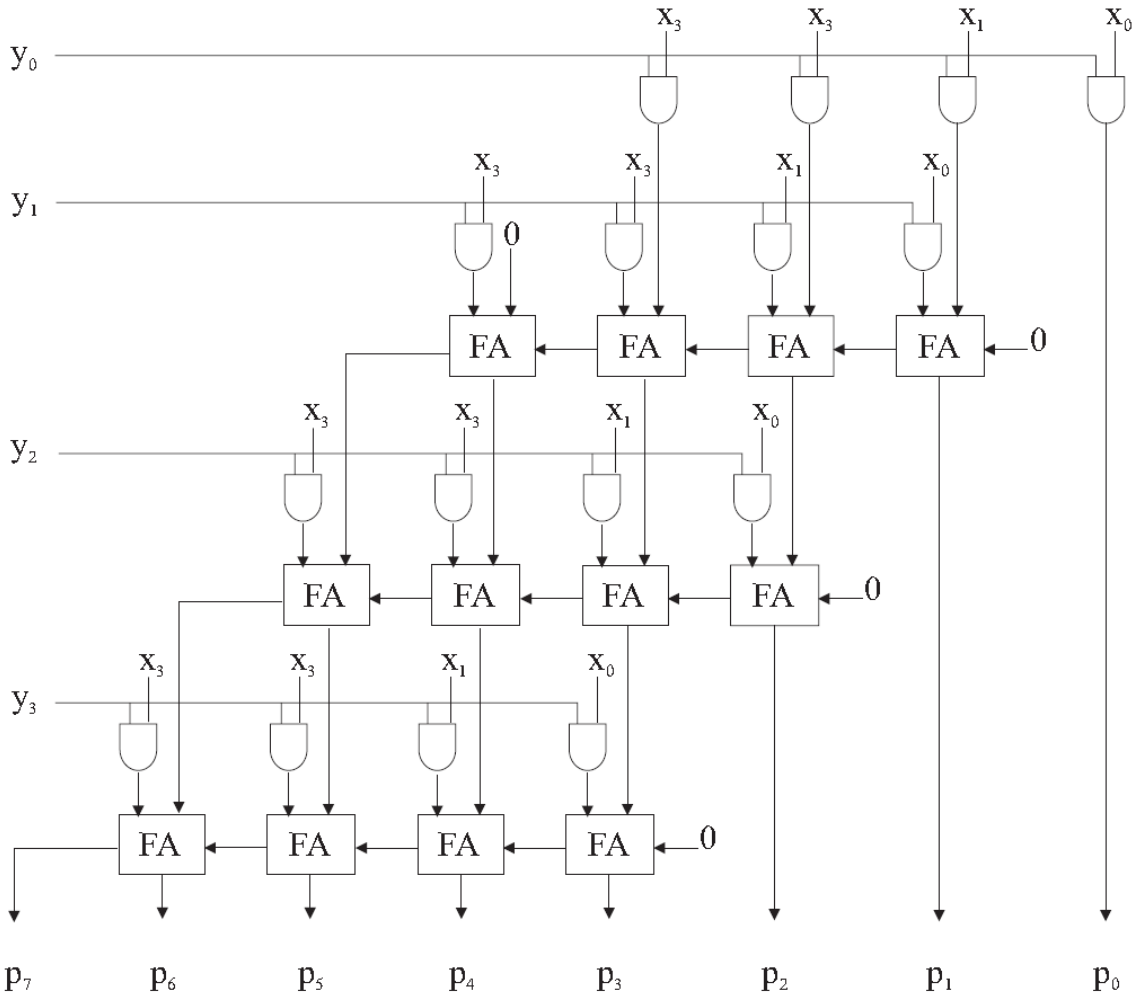


Figura 3-16 Multiplicador Ripple Carry de 2 operandos de 4 bits.

En la bibliografía disponible es usual analizar los esquemas de multiplicadores a partir de la síntesis de una unidad llamada procesador elemental (PE). Para el caso que nos ocupa, el procesador elemental consta de una puerta AND y un sumador completo, tal y como se muestra en la figura 3-17:

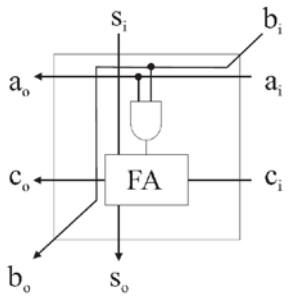


Figura 3-17 Procesador elemental de un multiplicador Ripple Carry.

Cada procesador elemental toma un bit de cada operando por las entradas a_i y b_i, calcula su producto a través de la puerta AND, suma el resultado proveniente de un PE

previo a través de s_i y el acarreo generado de un PE previo a través de c_i. El resultado de la suma a la salida es s_0 con el correspondiente acarreo c_0. Los operandos son transmitidos a la salida a través de a_0 y b_0.

Este multiplicador puede ser representado a partir de los procesadores elementales en el esquema mostrado en la figura 3-18:

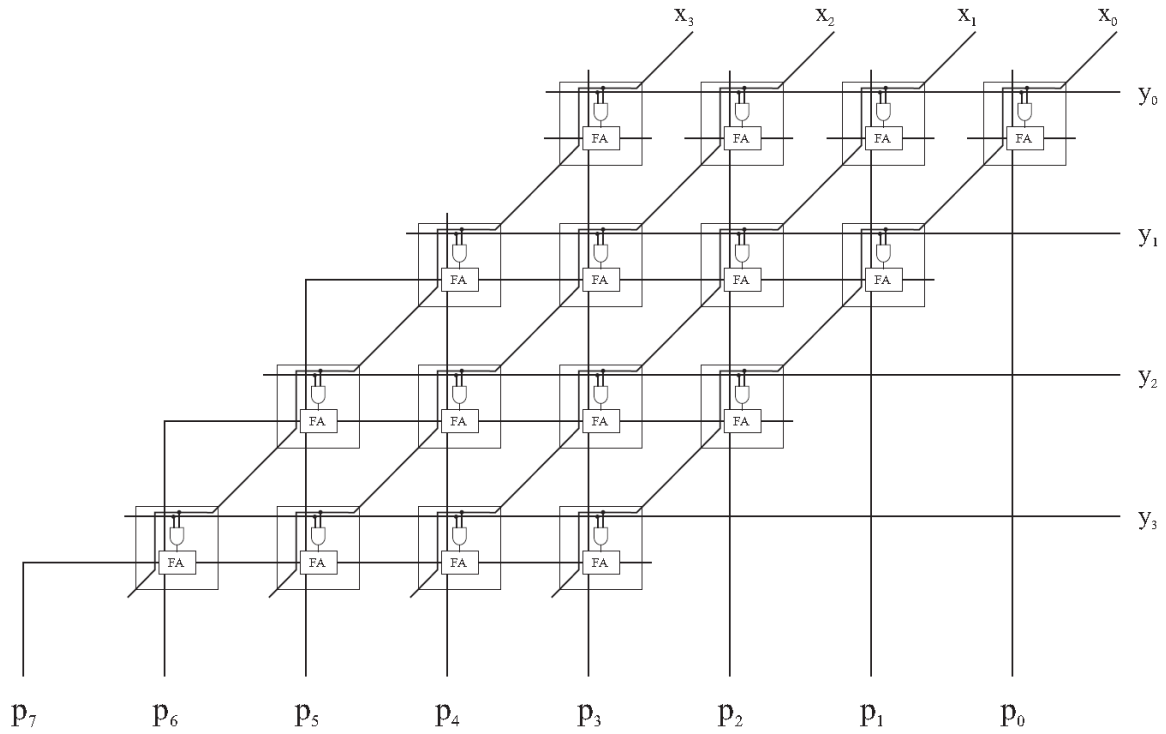


Figura 3-18 Multiplicador Ripple Carry mediante procesadores elementales.

3.3.2. Multiplicador Guild

En la figura 3-20 se muestra el multiplicador paralelo propuesto por H. Guild [34], cuyo procesador elemental es el de la figura 3-19.

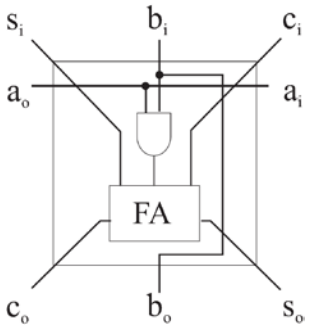


Figura 3-19 Procesador elemental de un multiplicador Paralelo Guild.

La estructura de este multiplicador se basa en cadenas de sumadores para cada bit del producto, las cuales se encuentran en dirección diagonal desde la esquina superior

izquierda hasta la esquina inferior derecha. Cada sumador de la cadena recibe un acarreo de entrada que corresponde a una suma de la cadena previa y transmite su salida de acarreo a un sumador de una cadena posterior.

Este multiplicador se caracteriza por una tasa alta de procesamiento, debido a que utiliza paralelismo y la posibilidad de implementación *pipeline*. Esta técnica consiste en una aceleración de procesos basada en la introducción de *latches* o registros en una determinada posición del array para separar la operación en etapas, y el sistema aumentará su velocidad de procesamiento tantas veces como etapas tenga.

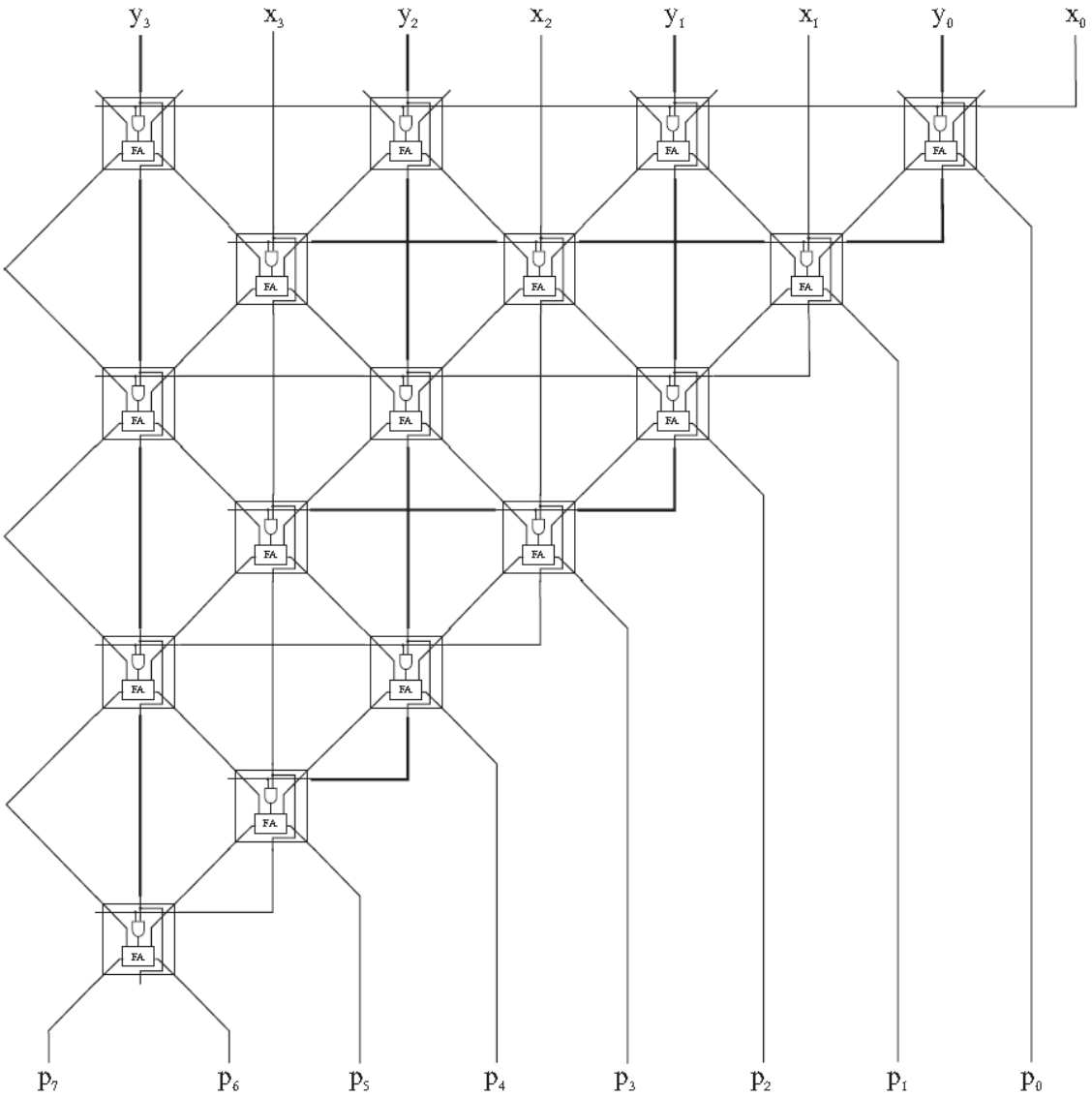


Figura 3-20 Multiplicador Paralelo Guild.

3.3.3. Multipicador McCanny-McWhinter

Otra propuesta de multiplicador paralelo es el de McCanny-McWhinter mostrados en las figuras 3-21 y 3-22, [35]. En la figura 3-21 puede observarse que este multiplicador está estructurado en cadenas de sumadores para cada bit del producto situadas en dirección vertical. Cada sumador de la cadena recibe un acarreo de entrada de forma diagonal proveniente de una suma de la cadena previa y transmite su salida de acarreo a un sumador de una cadena posterior. Este multiplicador tiene la característica de presentar comunicación local entre sus celdas básicas.

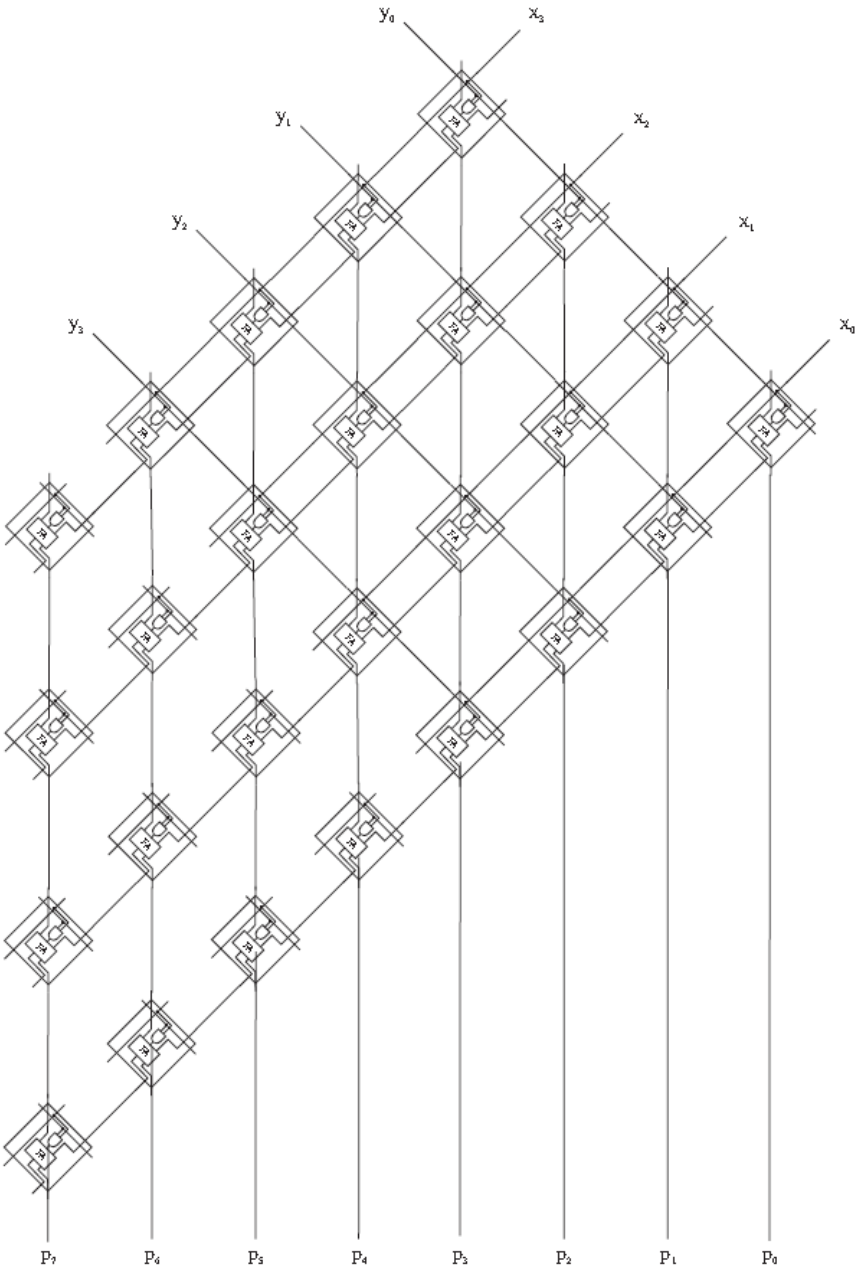


Figura 3-21 Multiplicador McCanny-McWhinter.

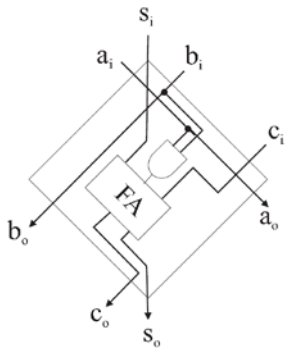


Figura 3-22 Procesador elemental del multiplicador McCanny-McWhinter.

3.3.4. Multiplicador por tabla de búsqueda (Look-up table, LUT)

Este multiplicador funciona como una memoria, no realiza cálculo. Se conectan los operandos X e Y constituyendo una dirección de memoria, cuyo contenido es el producto X·Y almacenado previamente. Este tipo de multiplicador es el más rápido, aunque depende de la velocidad de acceso a la memoria. Su gran desventaja es la cantidad de recursos lógicos que utiliza; por ejemplo, un multiplicador de 16 bits necesita una memoria de 4.294.967.296 x 32 bits.

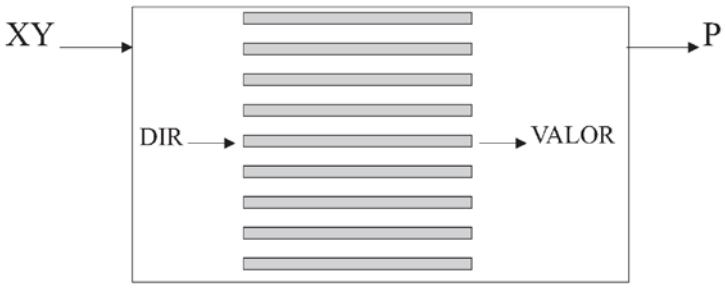


Figura 3-23 Multiplicador por tabla de búsqueda (LUT).

Este tipo de multiplicador se utilizará cuando realicemos el cálculo de la convolución en las FPGAs de Xilinx utilizando las *look-up table* de que disponen las FPGAs de bajo coste actuales que se describirán con mucho más detalle en el capítulo siguiente,

3.3.5. Multiplicador Carry Save

El multiplicador paralelo que será utilizado en la investigación de esta tesis es el multiplicador *carry–save*. Este tipo de multiplicador es generado con un esquema de propagación del acarreo como la suma *carry–save* descrita en el apartado 3.2.5. Este esquema trata de romper la cadena de acarreo para disminuir el retardo en cada suma, lo que nos permite acelerar la multiplicación.

La idea es romper la cadena de acarreo propagado del sumador *ripple* para disminuir el retardo de cada suma, lo cual permite acelerar la multiplicación. El multiplicador tiene la característica de permitir salvar el acarreo generado en las sumas parciales y transferirlo como acarreo de entrada a la siguiente suma parcial de la fila de productos parciales siguiente.

En las figuras 3-24, 3-25 y 3-26 se muestra este multiplicador, su procesador elemental y el multiplicador *carry–save* mediante procesadores elementales.

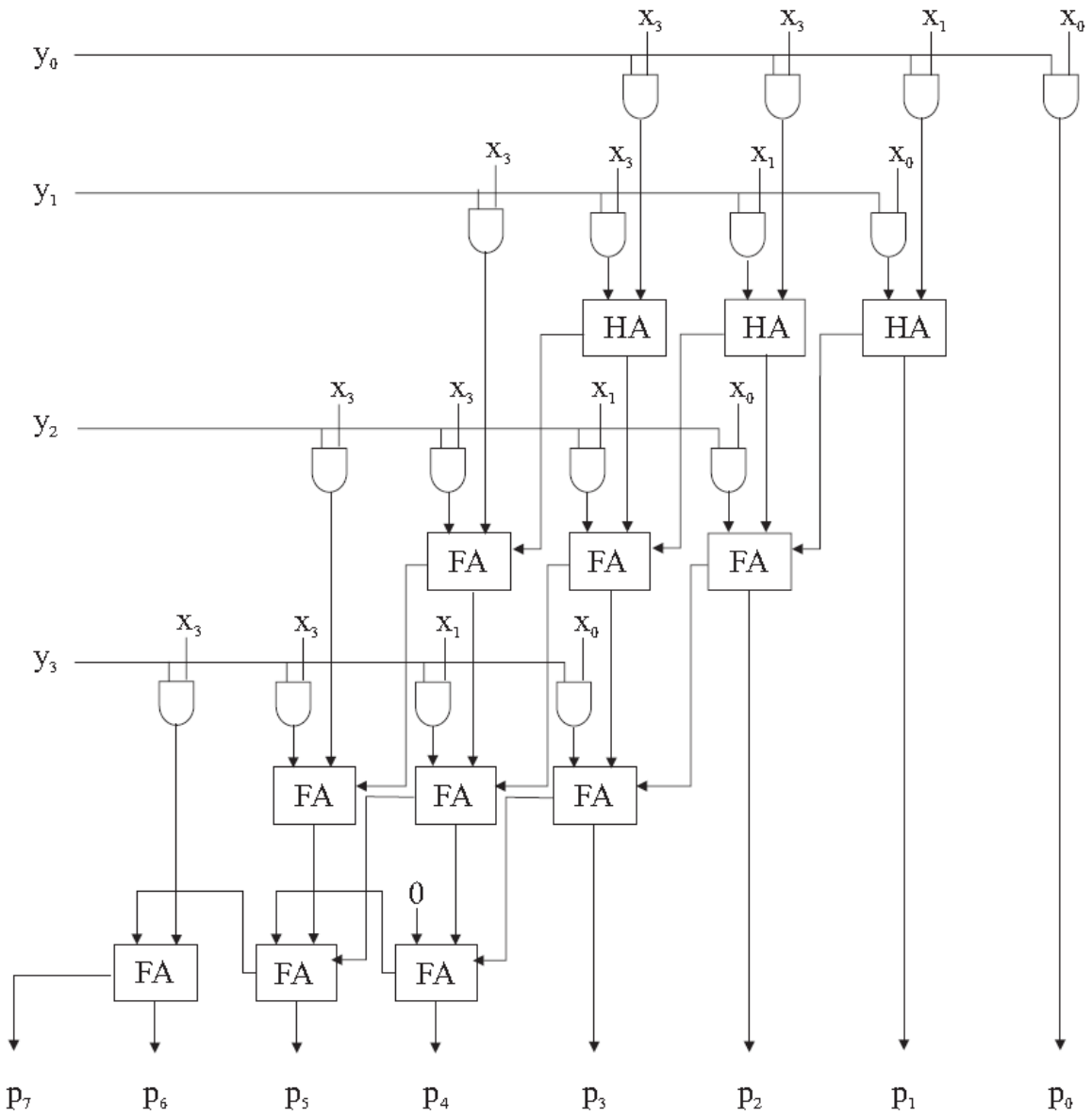


Figura 3-24 Multiplicador *carry–save* de 4x4 bits.

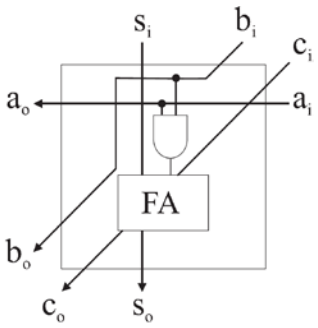


Figura 3-25 Procesador elemental de un multiplicador *carry save*.

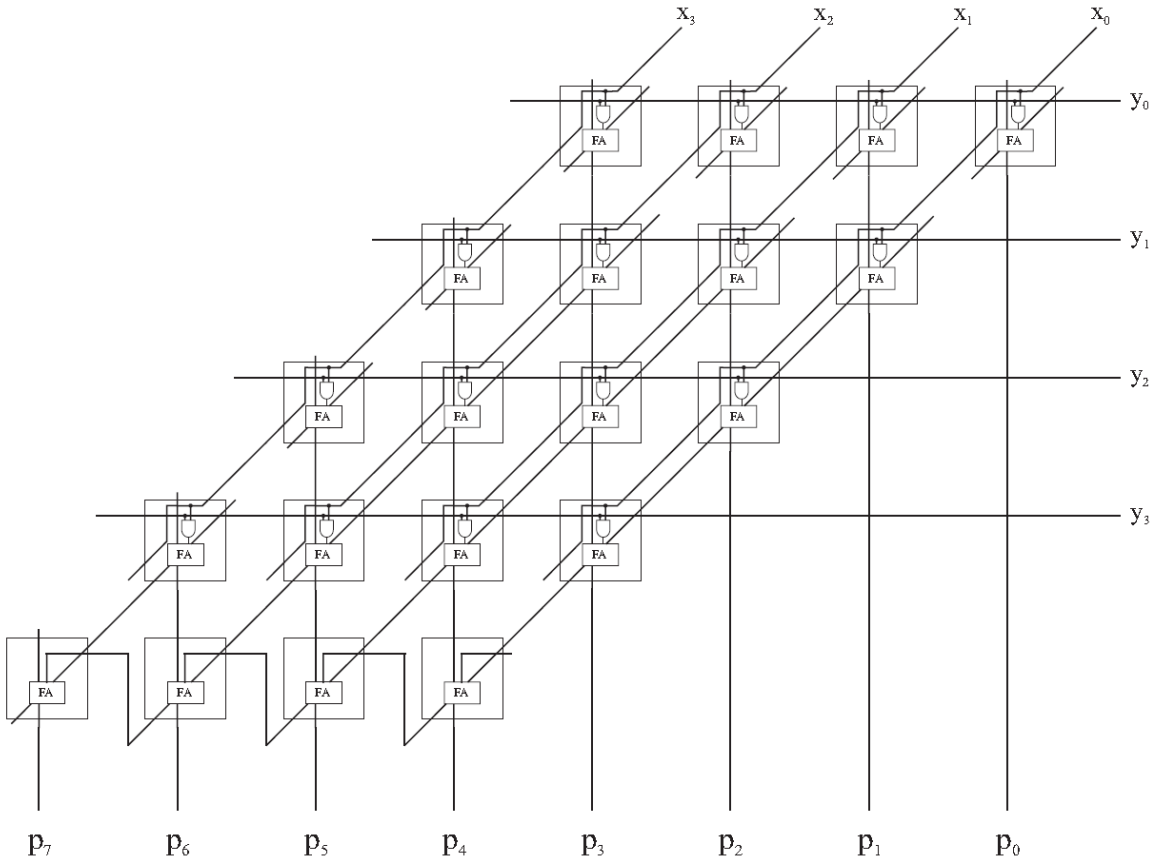


Figura 3-26 Esquema del multiplicador *carry save* mediante procesadores elementales.

Este multiplicador *carry-save* ha sido implementado en FPGA por Ortiz et al. en [36].

4 Circuitos electrónicos programables de bajo coste

RESUMEN

En el presente capítulo se realiza una descripción de los circuitos electrónicos disponibles en el mercado que se utilizan en el procesamiento digital de señales. Básicamente se estudian los procesadores digitales de señal (DSPs) y las FPGAs. En una primera instancia se describen con detalle las características de los procesadores digitales de señal (DSPs), para pasar posteriormente al estudio de los dispositivos lógicos programables (PLDs) comenzando con los simples pasando por los complejos (CPLDs) hasta situarnos en los dispositivos que se utilizan en esta tesis, que son las FPGAs (matriz de puertas programable en campo). Más en concreto se hace una descripción de la arquitectura de las FPGAs de Xilinx y sobre todo de los multiplicadores empotrados y bloques DSPs de los que disponen las FPGAs de bajo coste de las familias Spartan 3 y Spartan 6. El capítulo concluye con una comparativa de ambos tipos de circuitos electrónicos para justificar el uso de las FPGAs.

4.1. Introducción

En el Procesamiento Digital de Señales (PDS) se recurre, como operación elemental, al producto de dos operandos y la posterior adición de un tercero, estructura conocida como MAC (Multiplicador/Acumulador). Esta operación es básica para la convolución.

Los dispositivos de cálculo más empleados en el PDS son los DSPs (*Digital Signal Processors*). Si bien los DSPs son baratos y flexibles, suelen poseer pocas unidades MAC por lo que un proceso aritmético demanda la ejecución secuencial de las operaciones. Como consecuencia, una operación algorítmica consume un determinado tiempo de ejecución y, si fuera necesario utilizar un tiempo de ejecución menor, convendría adoptar hardware a medida, alternativa proporcionada por los ASICs (*Application–Specic*

Integrated Circuit) y las FPGAs. Tanto los ASICs como las FPGAs posibilitan la utilización de varias unidades MACs en paralelo, logrando de este modo una notable reducción del tiempo de procesamiento. Los ASICs admiten implementar sistemas complejos y resguardan la propiedad intelectual de los procesos debido a que no son copiables. Sin embargo implican un gran costo de fabricación por lo que no aceptan errores en su desarrollo y, consecuentemente, su utilización representa un gran riesgo de producción.

En cuanto a las FPGAs, éstas combinan la flexibilidad de un DSP con la velocidad y la densidad de componentes de un ASIC. Las FPGAs poseen una gran cantidad de recursos lógicos, un bajo costo de desarrollo ya que son fáciles de depurar y, fundamentalmente, permiten al diseñador corregir errores y actualizar el diseño. Estas ventajas convierten a las FPGAs en dispositivos apropiados como procesadores de señales o aceleradores de cálculo.

Durante la década de los 80, varias compañías intentaron resolver el compromiso de la complejidad de los circuitos y la estandarización de sus productos. Por un lado se tenía la opción barata del microprocesador que constituye un componente estándar, y por otro lado los ASIC denominados *full-custom*, es decir, circuitos integrados de aplicación específica diseñados completamente a medida, muy óptimos pero también muy caros. En estos, los ingenieros diseñaban todas las máscaras presentes en el proceso de fabricación de los circuitos integrados.

Por otro lado existía la posibilidad de realizar diseños denominados *semi-custom*, con las tecnologías denominadas *Gate Array* y *Standard Cells*. En la tecnología *Gate Arrays* se estandarizan las etapas iniciales de fabricación, por lo que los clientes comparten una estructura similar en forma de filas de transistores sin interconexión. La conexión se realizaba a medida en las últimas fases del proceso de fabricación, diseñando a medida una o más capas de metalizaciones, finalizando así el diseño. La tecnología *Standard Cells*, en cambio, no era un proceso prefabricado, pero se dispone de células prediseñadas y estandarizadas, como RAMs, ALUs, multiplicadores, etc., con lo que se optimiza en densidad, velocidad y área respecto los *Gate Arrays*.

Una de las formas de caracterizar las prestaciones de un diseño digital es evaluar conjuntamente el área que ocupa, la velocidad máxima a la que puede operar y la potencia

que consume, lo que se denomina figura área–tiempo–potencia (ATP). Teniendo esto en cuenta, los ASICs tienen mejores prestaciones que los componentes estándares como el microprocesador, además de ser más fiables y estar más protegidos. Por el contrario, los ASICs son mucho más difíciles de depurar y tienen un coste fijo elevado, por lo que necesitan un volumen de fabricación muy alto para ser rentables. Pero lo más desfavorable era, sin duda, la imposibilidad de corregir errores, ya que una vez se enviaba el circuito a fabricar, no se podían hacer modificaciones.

Los avances en la tecnología hacían que se pudieran elaborar sistemas cada vez más complejos, lo cual traía consigo la aparición de errores. Además, la propia complejidad de los sistemas hacía que su diseño implicara un tiempo de desarrollo cada vez mayor. Por otro lado, la ley de Moore hacía obsoleto cualquier producto en menos de dos años, lo cual implicaba una presión adicional sobre los ingenieros electrónicos.

Esta situación llevó a que algunos ingenieros buscaran ofrecer un producto, o una metodología de diseño que combinara el alto ATP de los ASICs con las principales ventajas de los microprocesadores: bajo costo, alta estandarización, facilidad para la corrección de los errores y reducción del tiempo de salida al mercado.

Varias compañías se lanzaron en los 80 a fabricar un ASIC que fuera reprogramable. La idea consistía en reemplazar la interconexión fija de los *Gate Arrays* y *Standard Cells* por una serie de pistas metálicas conectables por transistores de paso controlados por un conjunto de bits de control almacenados en una memoria interna.

4.2. Procesadores digitales de señal (DSPs)

En este apartado se pretende dar una introducción a la tecnología de los DSP, brindando un panorama general de las características que éstos tienen y donde se pueden implementar. Se pretende dejar claro que quizá la ventaja más grande de los sistemas de procesamiento digital de señales radica en la flexibilidad que tienen para implementar soluciones que integran algoritmos, software y hardware en un solo sistema, y modificarlos con tan solo cambiar unas líneas del programa. [37]

4.2.1. Pasado, presente, y futuro de los DSPs

En 1982, surgió el primer procesador digital de señal como tal. Con más de 30 años de evolución y existencia los DSP aumentan cada vez más su poder de cálculo, rapidez e

integración. No hay que pasar por alto el impacto de la tecnología en todo este tiempo. La industria ha usado 9.5 billones de DSP en diversos productos que han mejorado la vida del ser humano a un grado inimaginable para aquellos que ayudaron a desarrollar la era de los DSP. Si se piensa que estos últimos 30 años fueron una muestra, sólo hay que esperar y ver lo que pasará en los próximos 30 años. Los DSP están proliferando en nuevas áreas en el mercado de los semiconductores, demostrando que evolucionan para la innovación que cambiará la vida cotidiana.

La relación entre los procesadores digitales de señales con otras tecnologías como los aceleradores FPGA, han permitido sistemas en un solo encapsulado (*System On Chip*, SoC). La FPGA es un dispositivo electrónico programable de muy alta densidad, la cual se determina en base al número de puertas de que dispone, lo que los hace ideales para el diseño digital de circuitos. La habilidad de manejar la lógica a un nivel de puerta permite construir e implementar eficientemente una función deseada. Si simultáneamente se reserva al FPGA aquellas subfunciones de un algoritmo, el DSP puede concentrar su eficiencia en el desarrollo de las operaciones matemáticas. Esta innovación ha llevado a descubrimientos exponenciales en espacios de aplicación prácticamente impensables, tales como el entretenimiento, audio, video y comunicaciones, etc. Pero éstos son solo una muestra de las múltiples aplicaciones.

La década de los 90 se caracterizó por la introducción en masa de los DSP al mercado comercial. A partir de aquí se ha librado una extensa batalla entre fabricantes de DSP, vendedores de ASIC, desarrolladores de procesadores de propósito general y microcontroladores. Los procesadores de propósito general, no son una alternativa real para la implementación de tareas de procesamiento digital de señal, debido al costo y el consumo de energía. El microcontrolador (MCU, *MicroController Unit*) es una computadora montada en un circuito integrado que se usa para controlar dispositivos electrónicos. Este es un microprocesador que enfatiza la autosuficiencia y la efectividad en costo. Un microcontrolador típico contiene la memoria y las interfaces necesarias para una simple aplicación, mientras que un microprocesador de propósito general requiere de circuitos adicionales para proporcionar esas funciones. Los microcontroladores se encuentran en muchos equipos electrónicos, estos son una amplia mayoría del total de procesadores vendidos. Alrededor de 50% son simples controladores, mientras que un 20% han sido incorporados dentro de los DSP.

Los ASIC fueron desarrollados para funciones más especiales tal como decodificación MPEG, en la cual se han establecido estándares que permiten un largo número de aplicaciones que usan una misma función básica [38]. El problema de estos dispositivos es el largo ciclo de diseño y el alto costo de inversión para su desarrollo.

Los DSPs por su parte ofrecen una solución intermedia entre los ASIC y los procesadores de propósito general, permitiendo un rendimiento especializado y configurable por aplicación. A pesar de que esta tecnología ha sido primariamente implementada a sistemas comerciales, el potencial aplicativo en plataformas reconfigurables se está explorando cada vez más.

En un futuro próximo, el uso de los DSP será determinante, por ejemplo los sistemas de visión incorporados a los automóviles, permitirán que éstos puedan manejarse de manera autónoma; los sistemas de tele–conferencia podrán ser en tiempo real evitando así el retraso telefónico. En campos tan cotidianos como el hogar habrá cambios significativos; los sistemas de seguridad podrán dar un informe completo a través de cámaras que alertarán si algo extraño sucede, y será a través del reconocimiento de imágenes que se permitirá el acceso a las personas. Los sistemas GPS tendrán de manera similar tareas de la ubicación exacta de personas u objetos como por ejemplo el automóvil.

El enfoque de los próximos años es profundo, de manera que aumentarán los desafíos. Para traer portabilidad, conectividad e inteligencia a cada dispositivo electrónico, las compañías de semiconductores necesitarán ofrecer soluciones completas, que involucren hardware, software y herramientas de diseño en el procesamiento de señales. Con la especialización tecnológica y colaboración de la industria, los desarrollos tendrán que tener mejores rendimientos, precio y consumo de energía. [39]

4.2.2. Ventajas de los DSP

El procesamiento digital de señales ofrece diversas ventajas sobre los sistemas tradicionales analógicos. El más significativo es que los sistemas basados en DSP son capaces de lograr un menor costo en la implementación de algunas tareas que podrían ser difíciles o imposibles usando sistemas analógicos. También su versatilidad para cambiar valores o consignas en un programa los hace altamente competitivos. Ejemplos de dichas aplicaciones incluyen la síntesis y reconocimiento de lenguaje, *módems* de alta velocidad con corrección de error de código, etc. Todas estas tareas involucran una combinación de

control y procesamiento de señales (tomando decisiones respecto a bits o instrucciones recibidos). Otras ventajas adicionales sobre los sistemas analógicos se pueden encontrar en Lapsley [40] que son:

- **Estabilidad, repetitividad y comportamiento previsible.** En ocasiones se observa que en un sistema analógico algunas condiciones lo hacen variar, tales como cambios en la temperatura de los componentes, que modifican su valor; a esto se agregan las derivas por el envejecimiento de los componentes y las tolerancias de éstos. Otro factor a tomar en cuenta es el calibrar estos sistemas para que operen con precisión. La salida de un sistema de procesamiento digital de señal no depende de factores ambientales o variaciones en el valor nominal de los componentes, es posible obtener sistemas con una respuesta exacta, conocida y sin variación de un equipo a otro.

- **Tamaño.** El tamaño de los componentes analógicos varía con sus valores; por ejemplo, el tamaño de un condensador de 100µF difiere de uno de 10pF, en la implementación de un filtro analógico. En contraste, en un filtro digital implementado en DSP bastaría con programar sus coeficientes.

- **Funciones especiales.** Hoy en día, algunos de los DSP de última generación cuentan con periféricos apropiados para ejecutar algoritmos característicos de ciertas aplicaciones para el procesamiento de señales de audio, de imagen o video y que no podrían implementarse por medios analógicos.

- **Programabilidad.** El sistema DSP puede reprogramarse en campo para ejecutar nuevos algoritmos. En contraste, con los sistemas analógicos que requieren de diversos componentes físicos para realizar diferentes tareas.

- **Inmunidad al ruido en la transmisión y almacenamiento de la información.** Las señales digitales son mucho más inmunes al ruido que las analógicas, no obstante, también pueden verse afectadas por este

factor. En cuanto a capacidad de almacenamiento, esta es cada vez es mayor en menor área física del DSP.

Es conveniente tomar en consideración que para aplicaciones sencillas la complejidad de un sistema analógico es menor, lo que se traduce en un menor costo. Si lo que se necesita son aplicaciones de tiempo real para un muestreo de la señal, es necesario realizar todos los cálculos que requiere el algoritmo. Para señales de gran ancho de banda, la frecuencia de muestreo puede ser tan elevada que impida su cálculo mediante técnicas digitales.

4.2.3. Características Generales de los DSP

La estructura general que describe a un sistema para procesamiento digital de señales responde al diagrama de bloques que se muestra en la figura 4-1.

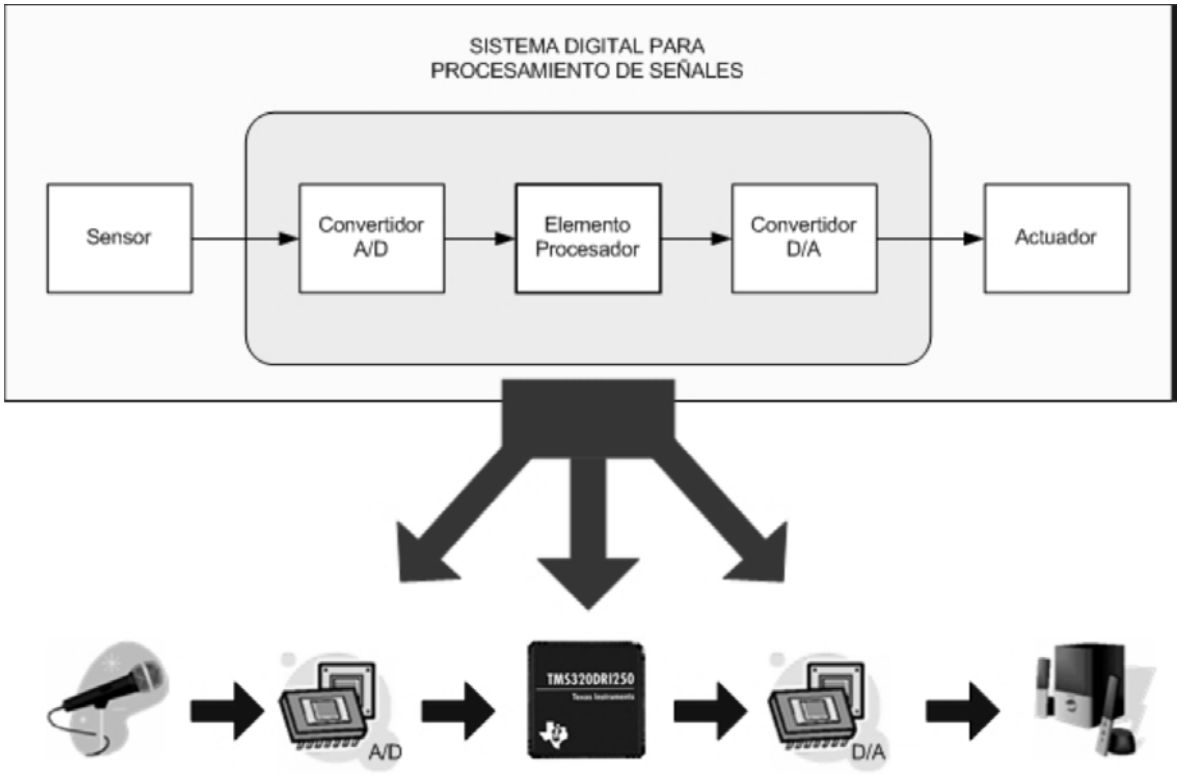


Figura 4-1 Configuración típica de un sistema de procesamiento digital.

La señal analógica se obtiene a través de un sensor que transforma una magnitud física en una señal eléctrica. Un convertidor analógico–digital convierte la señal analógica en una secuencia numérica. Estas muestras llegan a un elemento procesador en el que se ejecuta un algoritmo de procesamiento digital. La salida de este procesador se introduce a un

convertidor D/A para nuevamente obtener una señal analógica, la que a su vez se puede transformar en una magnitud física por medio de un actuador.

Un aspecto sumamente importante es que las características del elemento procesador estarán impuestas por los requerimientos de las aplicaciones en que se han de utilizar, y en la naturaleza de las operaciones que es preciso realizar de la señal. La operación básica es el producto acumulativo de dos secuencias, a la cual se le denomina operación **MAC** (*multiply and acumulate*), esta se manifiesta en aplicaciones como: convolución, filtrado, análisis espectral, correlaciones, etc. Los factores del producto pueden ser, dependiendo del caso, muestras de una señal, coeficientes de un filtro o constantes precalculadas como tablas de senos y cosenos.

En resumen, si se consideran las condiciones que impone la aplicación, las características generales que debe reunir el elemento procesador son:

- Deben ser dispositivos con una arquitectura que permita procesar las muestras de entrada a una gran velocidad, ya que en ocasiones son en tiempo real.
- Los algoritmos de procesamiento digital de señal tienen una elevada carga computacional en los que predomina la operación MAC, por lo que este elemento procesador debe tener una gran capacidad de cálculo.
- Dado el gran volumen de datos a procesar deben permitir un manejo accesible de los mismos.
- Tendrán que ser programables para hacer posible la implementación de distintos tipos de algoritmos de procesamiento digital de señal.

4.2.3.1. Algoritmos

Los sistemas de procesamiento digital de señales frecuentemente se caracterizan por la utilización de un algoritmo que especifica las operaciones aritméticas a realizar con los datos. La implementación del algoritmo se puede realizar mediante *software* en un microprocesador ordinario o un procesador de señal programable, o mediante *hardware* a la medida.

Para un sistema programado el algoritmo se implementa como un programa que se ejecuta secuencialmente, lo cual se traduce en:

- Una gran flexibilidad al poder utilizar un mismo sistema para implementar varios algoritmos.
- Los algoritmos pueden alcanzar un alto grado de complejidad.
- El hecho de ejecutar el programa en forma secuencial empeora el rendimiento.
- Existen aplicaciones donde se requiere procesamiento paralelo de señales.

Dentro de la implantación del algoritmo en software se encuentran dos alternativas. La primera de ellas utilizando microprocesadores de propósito general de altas prestaciones, como por ejemplo el Pentium de Intel, Power Pc 601 de Motorola e IBM, etc. Los sistemas que se basan en este tipo de procesadores son muy complejos porque usan una serie de periféricos y elementos externos para su implementación. De ahí que se conozcan como procesadores de propósito general. Además, estas altas prestaciones no siempre se traducen en una elevada potencia de cálculo o una rápida ejecución de operaciones tipo MAC, debido a las restricciones en la ejecución de operaciones en punto flotante.

La otra alternativa para la implementación del algoritmo en software son los procesadores digitales de señal, DSP, los cuales tienen una arquitectura específicamente diseñada para estas aplicaciones (arquitectura *Harvard*). Son dispositivos con prestaciones superiores y de menor costo que los de propósito general. Además, disponen de herramientas de desarrollo flexibles que permiten acelerar el proceso de diseño.

En lo que respecta a la implementación de algoritmos en hardware, ésta se realiza mediante circuitos integrados diseñados a la medida, integrando en el dispositivo sólo aquellas funciones necesarias para la aplicación concreta a que van destinados. Este *hardware* puede tomar múltiples formas. Una de ellas son los dispositivos estándar para aplicaciones específicas (ASSP, por sus siglas en inglés). Como su nombre indica, se trata de circuitos integrados que implementan un algoritmo concreto. Una ventaja es que pueden operar con señales de alta frecuencia, pero al mismo tiempo se dificulta el diseño del

dispositivo por lo que los algoritmos implementados son sencillos, de una estructura regular y con un número limitado de bits de operación. Un ejemplo de los ASSP son los filtros digitales programables, los cuales integran un conjunto de multiplicadores en hardware que operan en paralelo para implementar un filtro FIR.

Finalmente, los DSP reúnen una serie de elementos que los hacen diferentes del resto de los procesadores. Estas características se manifiestan en:

- La arquitectura Harvard de la CPU, que dispone de recursos que posibilitan las realizaciones de las operaciones MAC de forma rápida, y la utilización de modos de direccionamiento especiales para el manejo de *buffers*.

- Un juego de instrucciones optimizado para aplicaciones de tratamiento digital de señal, entre las que destacan aquellas que le permiten realizar un control eficiente de lazos.

- Una arquitectura de memoria que le permita obtener las instrucciones y datos a procesar a la velocidad que los demanda el CPU.

- La inclusión de un conjunto de periféricos en el mismo dispositivo que le permita comunicarse con el exterior.

4.2.3.2. Velocidad de Reloj

Los sistemas electrónicos digitales se caracterizan por su velocidad de reloj. La velocidad de reloj se refiere a la velocidad a la cual el sistema ejecuta sus unidades de trabajo más básicas. Para los sistemas de DSP, la relación de la velocidad del reloj y la velocidad de muestreo es una de las características más importantes usadas para determinar cómo será implementado el sistema. Es decir, esta relación determina parcialmente la cantidad de hardware necesario para implementar un algoritmo con una complejidad dada [38].

4.2.3.3. Velocidad de muestreo

Una característica clave de los sistemas DSP es su tasa de muestreo (*sample rate*). El muestreo es el proceso de convertir una señal en tiempo continuo a una señal en tiempo discreto, en intervalos de muestreo también discretos. Las amplitudes de las señales en

tiempo discreto se cuantifican en valores digitales con un ancho de palabra $N=2^n$ (donde n es el número de bits) dado. Un ADC realiza los procesos de muestreo y cuantificación de una señal como se observa en la figura 4-2.

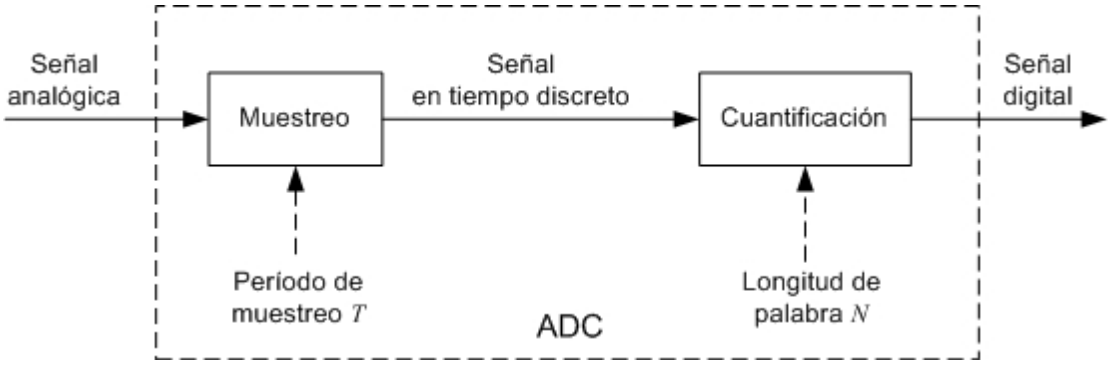


Figura 4-2 Proceso de conversión analógico-digital.

Es un error frecuente y extendido creer que una misma señal muestreada con una tasa elevada se reconstruye mejor que una que se muestrea con una tasa inferior. Esto es falso (siempre que las tasas empleadas cumplan el criterio de Nyquist, naturalmente). El proceso de muestreo (que no debe ser confundido con el de cuantificación) es, desde el punto de vista matemático perfectamente reversible, esto es, su reconstrucción es exacta, no aproximada. Dicho de otro modo, desde el punto de vista matemático al que se refiere el teorema de muestreo de Nyquist–Shannon, la reconstrucción de una señal de 10 kHz es idéntica tanto si se obtiene de una tasa de muestreo de 25000 muestras por segundo como de una de 50000 muestras por segundo. No aporta nada incrementar la tasa de muestreo una vez que esta cumple el criterio de Nyquist. Cuando la cantidad de señales muestreadas es excesivamente mayor se produce el efecto de sobremuestreo, donde no se aporta nueva información y por el contrario, hay mayor demanda de memoria RAM y mayor tiempo de procesamiento. Esto puede ser un fuerte inconveniente en aplicaciones de procesamiento en tiempo real.

Si se utiliza una frecuencia menor a la establecida por el teorema de Nyquist, se produce una distorsión conocida como traslape espectral (*aliasing*). El *aliasing* impide recuperar correctamente la señal cuando las muestras de ésta se obtienen a intervalos de tiempo demasiado largos. La forma de la onda recuperada presenta pendientes muy abruptas. Para ampliar los aspectos teóricos sobre el teorema del muestreo se pueden consultar el libro de Proakis & Manolakis [4].

4.2.3.4. Formatos de datos

Una de las características más importantes que determinan la idoneidad de un DSP para una aplicación dada, es el tipo de formato y número de bits de los datos con que realiza los cálculos matemáticos. Con relación al tipo de formato de datos, los DSP pueden operar con números en punto fijo, punto flotante o ambos.

4.2.3.4.1. Números en punto fijo

Algunos DSP solo son capaces de operar con números enteros, salvo que se considere la existencia de un punto binario, mediante el cual se escalan los valores enteros para de esta forma obtener números fraccionarios. Este factor de escala es igual a 2^{-bp} donde bp es la posición del punto binario. El DSP realiza las operaciones de suma o multiplicación como si se tratara de números enteros, sin considerar este factor de escala. Es responsabilidad del programador interpretar la posición del punto binario. En la figura 4-3 se encuentran algunos ejemplos de la representación de números en aritmética de punto fijo.

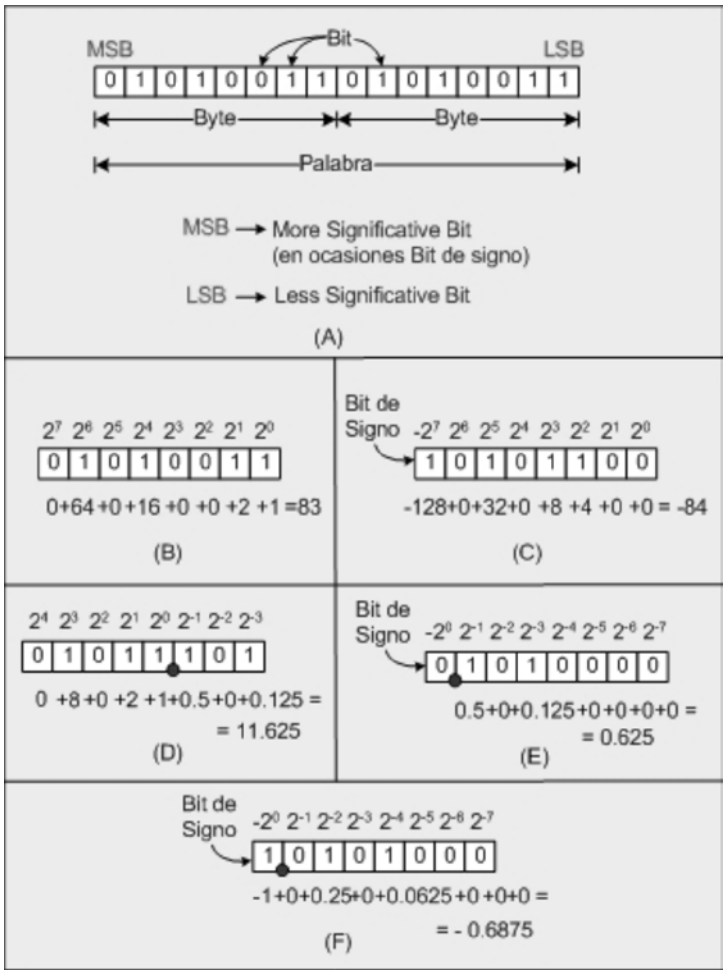


Figura 4-3 Representación de números en punto fijo. (A) Ancho de palabra. (B) Entero Positivo. (C) Entero Negativo. (D) Entero y fraccional. (E) Fraccional Positivo. (F) Fraccional Negativo.

Al desplazar el punto decimal a la izquierda, utilizando más bits para la parte fraccionaria, la precisión aumenta, pero disminuye el margen de valores de la representación. Puesto que el tamaño de la palabra de datos es fijo, la situación del punto binario será una situación de compromiso entre la precisión a obtener y el margen de valores a cubrir. El programador debe utilizar el mayor número de bits para la parte fraccionaria (máxima precisión) que permita representar todo el intervalo de valores que toma una variable.

Si durante el procesamiento, un número en punto fijo aumenta demasiado para poder ser representado con el número de bits disponibles para la parte entera, el programador debe realizar un escalado descendente del número mediante un desplazamiento a la derecha, perdiendo los bits de menor peso y por tanto disminuyendo la precisión. Si por el contrario el número en punto fijo disminuye demasiado, el número de bits utilizados en la parte fraccionaria puede ser insuficiente. El programador entonces deberá hacer un desplazamiento a la izquierda para aumentar la precisión.

En ambos casos el programador debe tomar en consideración como se ha ido desplazando el punto binario, restaurando todos los números de punto fraccionario a una misma escala en una etapa posterior. Esto convierte la programación en una tarea muy tediosa.

La utilización de rutinas en punto fijo que emulan las operaciones punto flotante (que permiten manejar cómodamente números fraccionarios) es muy costosa en cuanto a tiempo de ejecución del código, lo que hace imposible su uso en aplicaciones de tiempo real. [38]

4.2.3.4.2. Números en punto flotante

Otros procesadores disponen de un CPU capaz de operar directamente con números en punto flotante. La aritmética en punto flotante es un mecanismo más flexible que la de punto fijo, ya que con la primera los diseñadores tienen acceso a un intervalo de valores mucho más amplio y a una mejor precisión. Un ejemplo se ilustra en la figura 4-4. Esto facilita la programación, ya que no es necesario preocuparse por el escalamiento. Un número de punto flotante se representa mediante una mantisa y un exponente, siendo su $\text{Valor}=\text{mantisa}\cdot 2^{\text{exponente}}$.

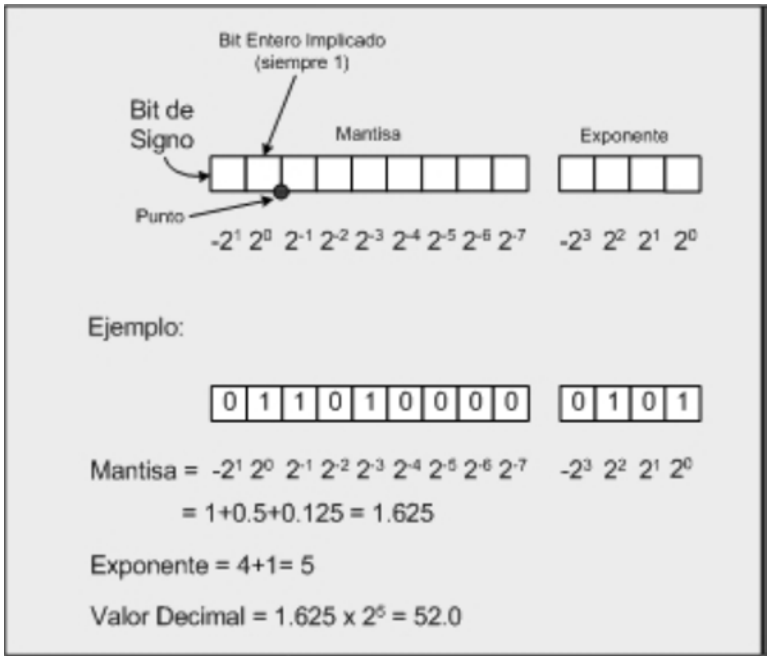


Figura 4-4 Representación de números en punto flotante.

La mantisa es un número fraccionario, mientras que el exponente determina la posición del punto binario. En estos procesadores es el propio hardware del CPU el que realiza los escalamientos mencionados anteriormente, quedando reflejada la posición del punto binario en el exponente. Esto facilita enormemente la programación.

4.2.3.5. Ancho de palabra de datos

Como se dijo anteriormente, la clasificación de los DSP se realiza con base en el tipo de aritmética que utilizan para realizar los cálculos matemáticos dividiéndose en DSP de punto fijo y DSP de punto flotante. Dentro de cada grupo se clasifican, además, según el ancho de su palabra de datos.

La CPU de los procesadores de punto fijo requiere un hardware más simple que el de punto flotante. Esto se traduce en una reducción del costo unitario del DSP, haciéndolos idóneos para aplicaciones de gran consumo que no requieran alta resolución. Esta simplicidad del CPU también reduce el consumo del dispositivo y su tamaño, un aspecto sumamente interesante para aplicaciones portátiles como los teléfonos móviles. La utilización de un CPU poco sofisticado permite liberar área del circuito integrado para incluir bancos internos de memoria RAM de mayor tamaño o incluso bancos EPROM o FLASH, donde grabar el código de la aplicación. Además, suelen disponer de un conjunto de periféricos más variado. De hecho, los procesadores destinados a aplicaciones

específicas (control de motores, sistemas de tratamiento de voz, etc.), son procesadores de punto fijo.

La anchura de la palabra de datos puede ser de 16, 24 o 32 bits. Esto tiene una importante repercusión en el costo, ya que influye poderosamente en el tamaño y número de terminales del dispositivo, y en los bancos de memoria externa conectados al mismo. Por lo tanto, los diseñadores intentan utilizar el circuito integrado con el menor ancho de palabra que su aplicación pueda tolerar.

Los DSP de punto flotante son dispositivos de escala alta, cuya CPU dispone de hardware específico para operar con datos de punto flotante. La anchura de la palabra con frecuencia es de 32 bits; si bien el uso de bits de guarda les permite operar en el interior del CPU con datos de 40 bits, esto no es obstáculo para que también puedan operar con datos en punto fijo. El área ocupada por la CPU en este tipo de DSP es mayor que en el caso de los de los de punto fijo. Por este motivo, la variedad de los periféricos que integran es menor, tratándose en la mayoría de los casos de periféricos (puertos serie y paralelo, DMA) utilizados en la comunicación con elementos externos (convertidores A/D y D/A). Los DSP más sofisticados disponen de puertos de comunicación que permiten el montaje de redes para un procesamiento en paralelo.

Cada tipo de procesador es ideal para un ámbito de aplicaciones. Los procesadores de 16 bits de punto fijo son adecuados para sistemas de voz, como teléfonos, ya que trabajan con el intervalo relativamente estrecho de las frecuencias de sonido. Las aplicaciones estéreo de alta fidelidad tienen un intervalo de frecuencias más amplio, de forma general, los requerimientos mínimos para este tipo de tareas serían un ADC de 16 bits y un procesador de 24 bits de punto fijo, de esta forma se proporciona un intervalo suficientemente amplio para obtener la señal de alta fidelidad y para poder manipular los valores que se obtienen al procesar la señal. El procesamiento de imágenes, gráficos en 3D y simulaciones científicas tienen un intervalo dinámico mucho mas amplio, por lo que se precisa de DSP de 32 bits con aritmética de punto flotante.

4.2.3.6. Paralelismo

Otra clasificación de los DSP es atendiendo al paralelismo de éstos, entendiendo como tal la posibilidad de ejecutar múltiples instrucciones de forma concurrente. Este paralelismo puede ser explícito o implícito. En la figura 4-5a se observa un DSP con paralelismo

explícito que integra varios CPU en un mismo encapsulado, los cuales se comunican entre si por medio de una memoria compartida interna. Esta opción se ha convertido en una vía muerta, ya que es responsabilidad del usuario el reparto del código a ejecutar por parte de cada uno de los CPU, lo que hace que sea programar las rutinas de comunicación una tarea muy tediosa.

Un DSP con paralelismo implícito como el de la figura 4-5b dispone de una sola CPU con múltiples unidades funcionales (por ejemplo, varias ALU, multiplicadores y conjuntos de registros) de forma que se puedan ejecutar en paralelo varias instrucciones, cada una de ellas sobre una unidad funcional. Para ello disponen de la palabra de instrucción de gran tamaño (*Very Long Instruction Word* de 256 bits), en la que se empaquetan varias instrucciones individuales. La planificación de qué instrucciones se van a ejecutar en paralelo en cada momento la realiza el propio compilador, en un proceso totalmente transparente para el programador. Este tipo de procesadores es sin duda la vanguardia de la tecnología de los DSP.

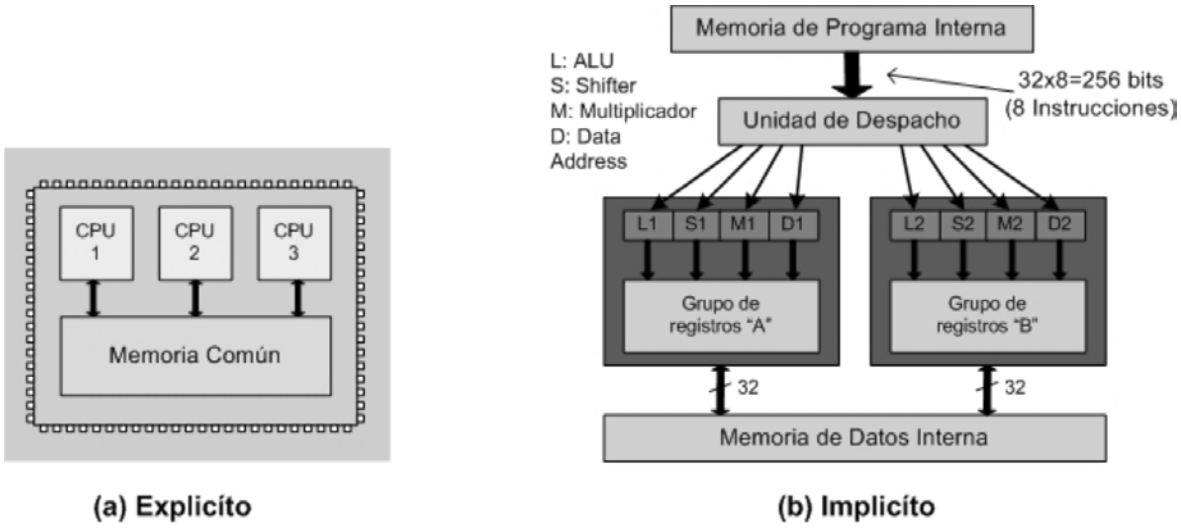


Figura 4-5 Tipos de DSP según su paralelismo

4.2.3.7. Arquitectura del DSP

Para incrementar la velocidad de trabajo y optimizar la implementación de los algoritmos de tratamiento digital de señal, la CPU de los DSP dispone de unidades computacionales específicas que pueden trabajar en paralelo. Entre ellas es básico contar con las unidades MAC, que posibilitan la ejecución repetitiva de este tipo de operaciones en un solo ciclo de instrucción.

Para lograr esta funcionalidad, los DSP incluyen un multiplicador y un acumulador integrado dentro de la ruta de datos (*data path*). Algunos autores se refieren a la ruta de datos como la unidad de procesamiento aritmética principal o ALU del procesador. La figura 4-6 muestra la ruta de datos típica, en este caso se trata del procesador de punto fijo DSP5600x de Motorola, de 24 bits. Muchos de los DSP del mercado tienen una arquitectura interna similar al del DSP5600x, donde las interconexiones y operaciones pueden variar.

A continuación se describen cada uno de los elementos básicos que conforman toda la ruta de datos, así como los multiplicadores y acumuladores.

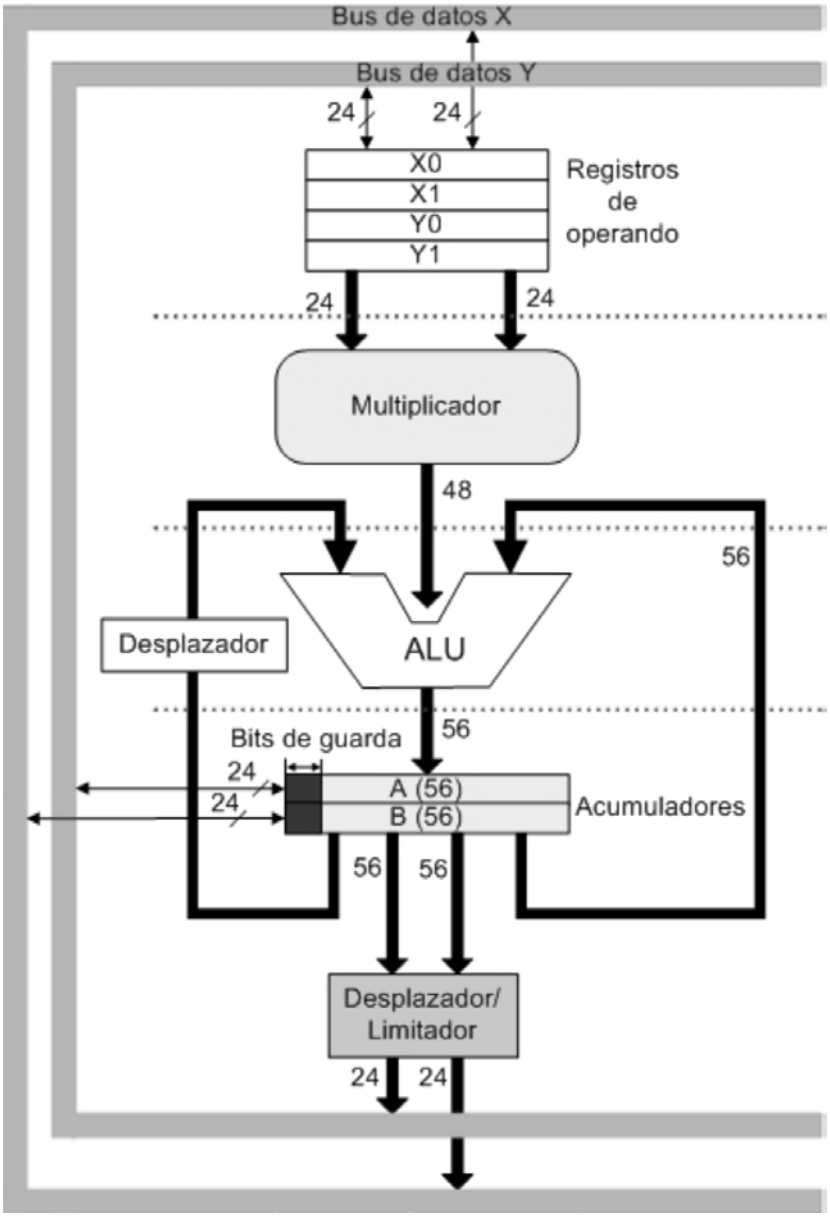


Figura 4-6 Ruta de datos representativo de un DSP de punto fijo (DSP5600x)

4.2.3.7.1. Multiplicador-Acumulador

La multiplicación es una operación esencial en casi todas las aplicaciones con DSP y es el multiplicador quien se encarga de realizar el producto de dos números en un solo ciclo máquina. Mientras el multiplicador produce un nuevo resultado por ciclo de instrucción, el *pipelining* interno del multiplicador puede tener un retardo de más de un ciclo, este retraso se denomina latencia.

En algunos casos el multiplicador se integra con un sumador, formando una unidad multiplicador–acumulador o **MAC**. Ambos elementos operan en paralelo para efectuar la operación MAC, siguiendo un proceso segmentado en el que, mientras el multiplicador realiza el producto de dos operandos la ALU acumula el resultado del producto anterior. Esta unidad MAC puede operar con números enteros o en punto flotante, y es la que interesa para la operación de la convolución planteada en esta tesis.

Al repetir las operaciones de acumulación de los productos puede ocurrir el desborde, obteniéndose resultados erróneos. Para evitar esta situación se sobredimensiona el número de bits de la ALU, añadiéndose un cierto número de bits extra, denominados bits de guarda, para realizar la suma de los términos producto. Por ejemplo, en un DSP que opera con datos de 24 bits el producto de dos números proporciona un resultado de 48 bits que se acumulará con los productos anteriores. Para evitar el desborde, se sobredimensiona la ALU añadiendo 8 bits de guarda, con lo que su tamaño final será de 56 bits. Este sobredimensionamiento de la ALU se aplica también al registro o registros donde se almacena el valor del acumulador.

4.2.3.7.2. Unidad Aritmético–Lógica (ALU)

La unidad aritmético lógica del DSP es la encargada de ejecutar las operaciones aritméticas básicas como suma, resta, incremento, etc. Además implementan operaciones lógicas tal como: AND, OR, y NOT. La estructura básica de una unidad aritmética lógica consiste en utilizar multiplexores con tantas entradas como operaciones realice dicha ALU.

4.2.3.7.3. Desplazador

Como resultado de la multiplicación y acumulación hay un aumento en la anchura de bits del resultado aritmético. En muchos de los casos, el programador escogerá un subgrupo

particular de los bits de resultado para pasarlos a la siguiente etapa de procesamiento. Un desplazador en la ruta de datos facilita esta selección escalando (multiplicando) su entrada por una potencia de dos (2^n). El desplazador se usa para preescalar un operando en la memoria de datos o el acumulador antes de una operación en la ALU, o también para postescalar el valor del acumulador antes de almacenarlo en la memoria de datos.

Una característica distintiva de los DSP es que sus desplazadores se implementan mediante lógica combinacional (denominada *barrel shifter*) frente a la estructura clásica de un registro de desplazamiento secuencial de otros tipos de procesadores. Esto permite realizar desplazamientos en un solo ciclo de instrucción, independientemente del número de bits a desplazar. Algunos procesadores tienen múltiples desplazadores con diversas capacidades en diferentes lugares de la ruta de datos. Esto permite una mayor flexibilidad para realizar desplazamientos en los lugares en que se está implementando cada algoritmo. Por ejemplo, el DSP5600x de la figura 4-6 tiene dos desplazadores independientes: uno se usa para escalar los resultados de multiplicar–acumular del acumulador a memoria, y el otro se usa para cambiar los valores dentro del acumulador, incluyendo lógica y diferentes tipos de rotación.

4.2.3.7.4. Desborde y saturación

Muchas aplicaciones de DSP involucran la acumulación en series de valores. Esto sucede, por ejemplo, en algoritmos de convolución y filtrado, cuando los elementos de una serie de datos se multiplican por coeficientes y los productos resultantes se suman. Una situación de desborde se manifiesta cuando al sumar dos números positivos se obtiene un resultado negativo y viceversa. Para solucionar esto, los DSP incorporan técnicas de saturación aritmética agregando circuitos que al detectar una situación de desborde durante la suma de dos números positivos, sustituyen la salida errónea por el máximo valor positivo a representar.

De igual manera, si la situación de desborde se da durante la suma de dos valores negativos proporciona como salida el máximo valor negativo a representar. El resultado sigue siendo erróneo, pero el error cometido en la operación es menor que en caso de desborde. De esta forma se consigue un comportamiento similar al de un circuito analógico en saturación. La saturación aritmética se puede realizar mediante una instrucción especial

o automáticamente. Algunos fabricantes denominan limitador al módulo que implementa la saturación aritmética. [38]

4.2.3.7.5. Unidad Generadora de Direcciones de Dato (DAG)

Otra característica que permite aumentar la velocidad de procesamiento aritmético en los DSP es la disponibilidad de una o más unidades generadoras de direcciones de datos (DAG, *Data Address Generator*), que calculan la nueva dirección necesaria para el acceso a los operandos. Estas unidades constan de una unidad aritmética específica que opera en paralelo con el resto de las unidades funcionales, y de un conjunto de registros que proporcionarán la dirección base y el desplazamiento necesario para el cálculo de la nueva dirección.

Muchas de las aplicaciones de procesamiento digital de señal necesitan gestionar un *buffer* con estructura FIFO (*First In, First Out*), donde se van almacenando las muestras que llegan del exterior o de cálculos anteriores. En la gestión del movimiento de los datos dentro y fuera del *buffer*, el programador mantiene un puntero de lectura y otro de escritura, los cuales se suelen almacenar en registros de direcciones. El puntero de lectura apunta a la posición de memoria que almacena el próximo dato que se va a leer del *buffer*, mientras que el puntero de escritura apunta a la posición de memoria donde se va a almacenar el siguiente dato de entrada. Cada vez que se realiza una operación de lectura o escritura, el puntero correspondiente avanza una posición, debiendo comprobar el programador si dicho puntero ha llegado a la última posición de memoria del *buffer*, inicializándose en tal caso para que apunte al comienzo del *buffer*. Esta comprobación de si un puntero ha llegado al final del *buffer* y su inicialización en caso afirmativo consume tiempo.

Para resolver este problema, las unidades DAG de muchos DSP realizan de forma automática la comprobación del puntero del *buffer* y su ajuste relativo a la dirección de comienzo, si es preciso, después de cada cálculo de dirección. Esta característica se denomina aritmética modular, la cual hace referencia a que el resultado de salida se limita a un determinado intervalo, de manera que si una operación de suma o resta supera tal rango, se obtendría el valor inicial de dicho intervalo.

Finalmente, cabe señalar que las DAG de varios DSP permiten implementar modos de direccionamiento pensados especialmente para la realización rápida de la transformada

discreta de Fourier. A este modo de direccionamiento se le denomina *bit–reverse* o de inversión de bits.

4.2.3.8. Repertorio de Instrucciones

El repertorio de instrucciones es un factor clave a la hora de determinar no sólo qué operaciones son posibles en un procesador, sino también cuando su uso es natural y eficiente. Las instrucciones controlan y operan los datos del CPU, cómo se leen y almacenan éstos en memoria, etc.

Las operaciones a realizar en el procesamiento digital de señal son poco variadas, y la mayoría de las veces se limitan a multiplicaciones y sumas. Los DSP disponen de un juego de instrucciones optimizado para este tipo de aplicaciones, con un número de instrucciones reducido, implementándose frecuentemente como microprocesadores con arquitectura RISC. Los beneficios de estas instrucciones especiales son dobles: por una parte permiten un código más compacto que requiere menos espacio en memoria; por otra parte incrementan la velocidad de ejecución de algoritmos específicos del procesamiento de la señal.

Estas instrucciones están optimizadas para las aplicaciones mencionadas, haciendo uso del paralelismo interno del CPU. Así, una misma instrucción puede realizar el producto de dos números almacenados en las posiciones de memoria especificadas por respectivos registros de direcciones, acumular el resultado con el contenido de otro registro de datos e incrementar los registros de direcciones.

Los algoritmos de DSP más frecuentes (convolución, correlación, multiplicación de matrices, etc.) se realizan mediante la ejecución repetitiva de una misma instrucción o conjunto de instrucciones denominadas lazos internos o núcleos de algoritmo.

La penalización en la ejecución que introducen las instrucciones de salto es especialmente importante en núcleos pequeños. Puesto que la mayoría de estos bucles se ejecutan en un número fijo de veces, el procesador debe utilizar un registro para almacenar el índice del lazo. La CPU debe utilizarse para incrementar el índice y comprobar si se verifica la condición de repetición del bucle. Si es así, se vuelve al comienzo del lazo mediante una instrucción de salto condicional. Todos estos pasos penalizan la ejecución del lazo y utilizan registros innecesariamente.

Para solventar estos problemas, los DSP utilizan lazos *hardware* (*zero overhead looping*). Éstos son estructuras de control especiales que repiten una sola instrucción, o conjunto de instrucciones, un determinado número de veces. Son especialmente efectivos para el caso de lazos de instrucción única, ya que se precisa traer la instrucción de memoria una única vez, liberando los buses para realizar otras operaciones como el acceso a datos o coeficientes. La diferencia fundamental con los lazos software es que un lazo hardware no pierde tiempo en incrementar o disminuir un registro, comprobar si se ha llegado al final del lazo o saltar al origen de éste.

4.2.3.9. Arquitectura de Memoria

En las aplicaciones de procesamiento digital de señal se procesa un gran volumen de datos. Debido a la limitación del número de registros disponibles en el CPU, normalmente estos datos residen en memoria. Para poder realizar las operaciones MAC en un solo ciclo de instrucción es preciso que los operandos utilizados estén disponibles en el momento de ejecutar dicha instrucción. Normalmente la acumulación de productos se realiza sobre un registro del CPU, con lo que el almacenamiento del resultado final en memoria no se considera parte del núcleo del algoritmo. Es decir, la ejecución de una instrucción MAC implica la realización de tres accesos a memoria:

- Un primer acceso de búsqueda de código de la instrucción a ejecutar.
- Dos accesos para la lectura de los operandos del producto

Teniendo en cuenta estas condiciones, los DSP estructuran su memoria de forma que se pueda obtener la instrucción a ejecutar y sus operandos desde memoria al ritmo que los demanda la CPU. De hecho, una de las características distintivas de los DSP es la forma en que organizan la memoria, siendo ésta un factor crítico en las prestaciones del procesador. Dicha arquitectura está orientada a posibilitar la realización simultánea, en un solo ciclo de instrucción, de los tres accesos anteriormente mencionados.

4.2.3.9.1. Arquitectura Von Neumann

La arquitectura de memoria utilizada por la mayoría de los microprocesadores convencionales, responde al modelo clásico de la arquitectura *Von–Neuman*, según la figura 4-7. Este modelo, dispone de un único espacio de memoria en el que se almacena tanto el código a ejecutar como los datos del programa. Los accesos a este espacio de

memoria se realizan a través de un único grupo de buses (bus de direcciones y bus de datos). Si bien se trata de una estructura muy simple, el procesador sólo puede realizar un acceso (de lectura o escritura) a memoria durante cada ciclo de instrucción.

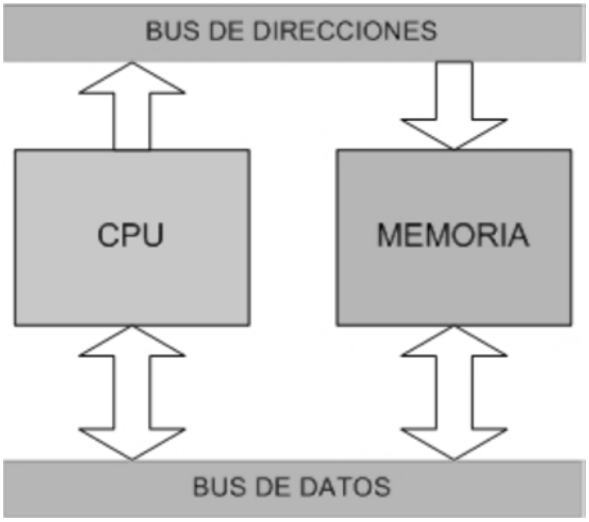


Figura 4-7 Arquitectura Von-Neumann.

En esta situación aunque la CPU del procesador fuera capaz de ejecutar una instrucción MAC en un solo ciclo de instrucción, como es el caso de los DSP, se requerirían tres accesos secuénciales a memoria para completar su ejecución. Puesto que la duración de cada acceso es de un ciclo instrucción, su ejecución se prolongaría durante tres ciclos.

Esta es una de las razones por la que los procesadores convencionales, aun teniendo una elevada potencia de cálculo, no son los más indicados para la ejecución de algoritmos de procesamiento digital de señal.

4.2.3.9.2. Arquitectura Harvard

La arquitectura Harvard, que se muestra en la figura 4-8, es una mejora que aumenta el ancho de banda de los accesos, aumentando el paralelismo de la memoria. Para ello se dispone de dos espacios de memoria independientes, uno para almacenar el código a ejecutar y el otro para los datos. Cada uno de estos espacios dispone de su propio grupo de buses, con lo cual es posible acceder simultáneamente a ambos espacios. La arquitectura Harvard original restringe el uso a que se destina cada uno de los espacios de memoria (almacenamiento de código o de datos). Esta solución no es del todo adecuada, ya que en instrucciones de tipo MAC es preciso acceder a dos operados en memoria, manteniéndose por tanto el cuello de botella en los accesos a memoria de datos.

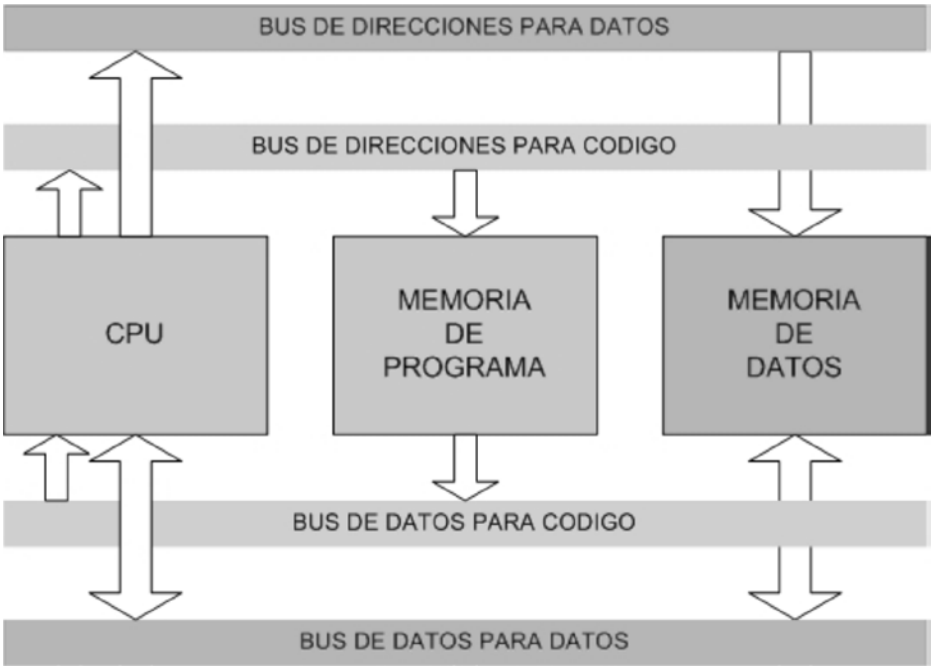


Figura 4-8 Arquitectura Harvard

En la figura 4-9, esta restricción inicial se ha flexibilizado dando lugar a la arquitectura Harvard modificada, en la cual se permite el almacenamiento de código y datos en el espacio de memoria de programa. Puesto que los DSP suelen disponer de módulos de repetición de instrucciones, una vez obtenido el código de la instrucción MAC, el espacio de memoria de programa queda disponible para obtener uno de los operandos. Por tanto, a excepción de la primera vez que se ejecuta dicha instrucción, las restantes completan su ejecución en un solo ciclo de instrucción.

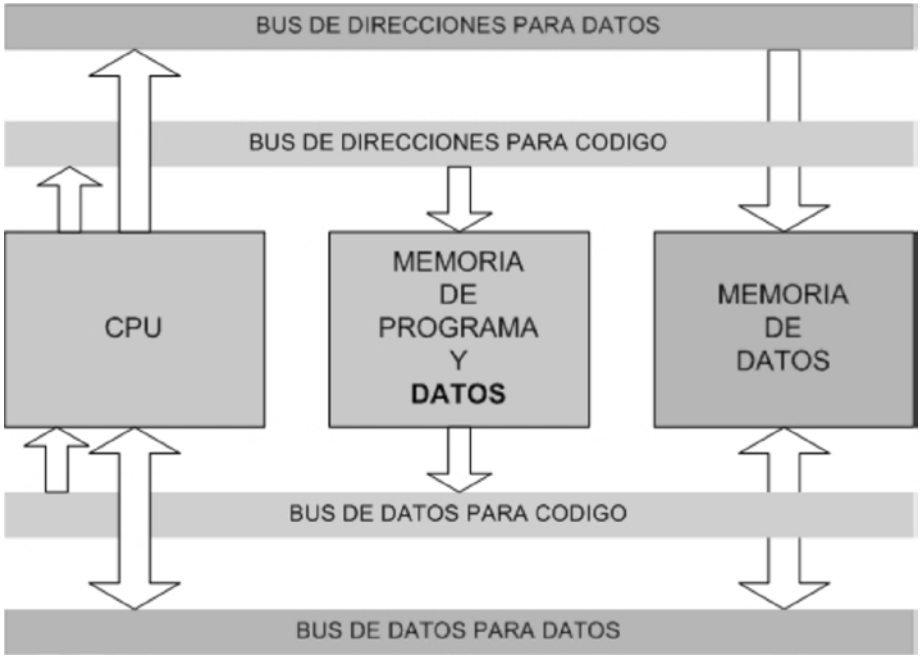


Figura 4-9 Arquitectura Hardvard modificada

Otros enfoques eliminan la latencia introducida durante la ejecución de la primera instrucción MAC desarrollando el concepto de la arquitectura Harvard. Para ello dividen a su vez la memoria de datos en dos bancos de memoria, cada uno de los cuales dispone de su propio juego de buses de datos y direcciones. Así se dispone de un banco de memoria de programa y dos bancos de memoria para datos, denominados X e Y, respectivamente.

Estas tres memorias permiten al procesador realizar tres accesos independientes por cada ciclo de instrucción (figura 4-10):

- Una búsqueda de instrucción.
- Un acceso de datos al banco X.
- Un acceso de datos al banco Y.

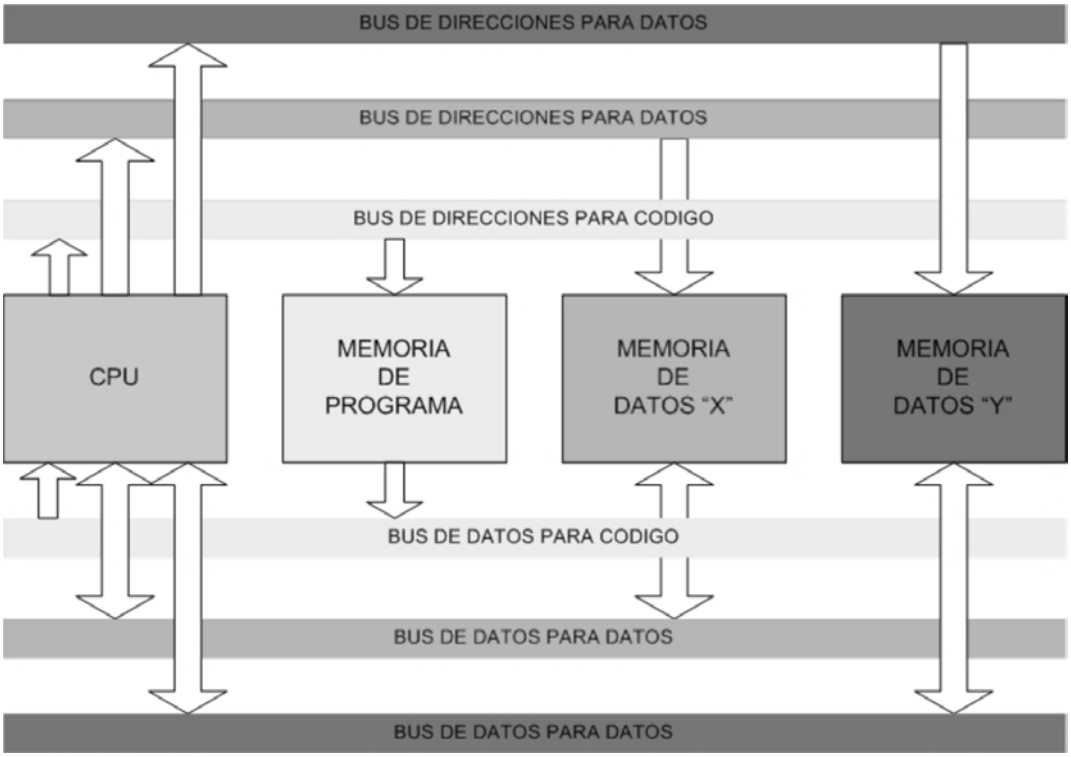


Figura 4-10 Arquitectura Harvard mejorada

En el caso de los DSP más recientes, desaparecen las restricciones referentes al uso al que se destina cada uno de los espacios de memoria. En su lugar se dispone de un único espacio compuesto por varios bancos de memoria independientes, cada uno con su propio grupo de buses. Es responsabilidad del programador distribuir cada una de las secciones del programa sobre distintos bancos para aprovechar al máximo el paralelismo del procesador.

Puesto que el aumento de múltiples buses de memoria fuera del circuito integrado es costoso, los DSP generalmente proporcionan un único conjunto de buses externo. Los procesadores con múltiples bancos de memoria normalmente proporcionan una pequeña cantidad de memoria interna para cada uno de los bancos. Aunque los bancos de memoria pueden ampliarse externamente, no se pueden realizar accesos externos múltiples en paralelo, debido a la ausencia de un segundo grupo de buses para la memoria externa. Por tanto, si se precisan múltiples accesos a memoria externa durante la ejecución de una instrucción, la ejecución de ésta se prolongaría a lo largo de varios ciclos de instrucción ya que los accesos a memoria se realizarían de forma secuencial.

4.2.3.9.3. Memorias de Acceso Múltiple

Existen otras alternativas no excluyentes al empleo de la arquitectura Harvard, que permiten aumentar el ancho de banda de la memoria. Una de ellas se basa en el uso de memorias de acceso múltiple. Se trata de memorias lo suficientemente rápidas para permitir varios accesos secuenciales por cada ciclo de instrucción a través de un único grupo de buses, o bien utilizar memorias multipuerto que permitan varios accesos concurrentes a memoria sobre dos o más grupos de buses independientes, como se muestra en la figura 4-11.

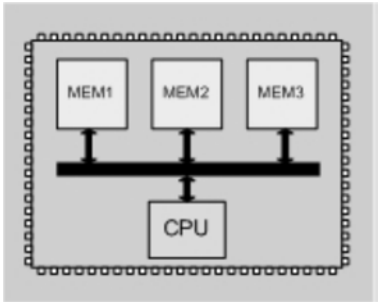


Figura 4-11 DSP con varios bancos de memoria internos

Tanto las memorias rápidas como las multipuerto se integran en el propio encapsulado del DSP. En el caso de las memorias rápidas situarlas en el exterior implicaría la introducción de retardos adicionales que podrían imposibilitar la realización de varios accesos en un solo ciclo; mientras que para las memorias multipuerto, su situación en el exterior significaría elevar el número de terminales de E/S del DSP, lo que incrementaría su costo y tamaño. Independientemente del tipo de implementación, la capacidad de los bancos de memoria de acceso múltiple suele ser reducida debido a la limitación del área disponible del circuito integrado.

Para completar en un solo ciclo la ejecución de la instrucción MAC (incluso el almacenamiento en memoria del resultado de la acumulación) es preciso distribuir adecuadamente las distintas partes del programa (código, operandos y resultados) de forma que no se sobrepase el ancho de banda de cada banco. En caso contrario, la ejecución del programa se volvería más lenta. Finalmente, otra de las alternativas para aumentar el ancho de banda de la memoria consiste en el uso de memorias caché de programa, las cuales son pequeñas memorias que se agregan dentro del núcleo del procesador, y que reducen la necesidad de acceder a memoria en la fase de búsqueda de instrucciones. Eliminar este acceso permite un posible acceso que se empleará para leer o escribir un dato, o bien puede eliminar los retardos asociados con una memoria de programa externa más lenta.

Respecto a la interfaz externa con memoria de un DSP, ésta puede caracterizarse básicamente mediante tres propiedades: el número de puertos disponibles, la sofisticación y flexibilidad de los mismos, y los requerimientos temporales.

Aunque cuenten con varios bancos independientes de memoria internos, la mayoría de los DSP disponen de un único conjunto de buses externo. Es así porque extender los buses al exterior implica encapsulados con un número de terminales muy elevado, lo que incrementaría notablemente el precio final. Por ello, es imposible realizar varios accesos a posiciones externas en un mismo ciclo de instrucción. La duplicidad de buses externos solo aparece en dispositivos de alta escala.

Las interfaces con memoria externa de los DSP varían bastante en cuanto a sofisticación y flexibilidad. Algunos son relativamente simples, con tan solo unas pocas terminales de control.

Otros son mucho más complejos, permitiendo conectar un mayor abanico de dispositivos de memoria externa y buses, sin necesidad de hardware adicional. Algunas características que distinguen a estas interfaces son la flexibilidad y concentración de los estados de espera programables, la disponibilidad de facilidades para implementar entornos multiprocesador (terminales para petición y cesión de bus, selección de acceso a memoria compartida, accesos con ínterbloqueo para implementar semáforos, etc.) y soporte para memorias DRAM en modo página o SRAM síncronas.

No suelen permitirse accesos múltiples a memoria externa en un solo ciclo de instrucción. Es más, la temporización varía dependiendo del sentido de la transferencia. Si

bien los accesos en modo lectura se pueden completar en un solo ciclo, las escrituras suelen prolongarse durante dos o más ciclos. Por este motivo la memoria interna se suele reservar para almacenar las partes del programa que se pueden modificar durante la ejecución de éste (pila y variables), mientras que el código se almacena en memoria externa.

4.2.3.10. Periféricos integrados e Interfaces de I/O

La mayor parte de los DSP integran en el propio encapsulado periféricos de gran versatilidad que le permiten comunicarse con dispositivos externos, tales como un ADC, un DAC, otros procesadores DSP o microprocesadores. Este es un aspecto de gran importancia y que los distingue de los procesadores de propósito general. De entre los más frecuentes se pueden encontrar:

- **Temporizadores**. Normalmente todos los procesadores incluyen estos dispositivos, que suelen admitir como entrada un reloj interno o externo. Se utilizan para la generación de interrupciones periódicas que sincronizan el proceso de muestreo.

- **Puertos Serie**. Las características y complejidad de estos interfaz varían de un procesador a otro, pero es habitual encontrarlos en la mayoría de ellos. Suelen utilizarse para comunicarse con circuitos que incluyen conversores A/D y D/A denominados "*codecs*". Esto permite alejar la parte analógica del procesador, eliminando problemas de ruido. De hecho, la mayoría de los fabricantes de los DSP diseñan dispositivos de este tipo con una interfaz que permite la conexión directa con el puerto serie. No es raro encontrar también DSP que permiten la transmisión del código de arranque en sistemas sin memoria ROM.

- **Puerto Paralelo**. Un puerto paralelo recibe y envía múltiples datos de entre 8 y 16 bits al mismo tiempo. Estos trasmiten la información mucho más rápido que un puerto serie; el inconveniente es que se requieren más pines para esta acción. Para ahorrar pines en la configuración de un puerto paralelo de comunicación con dispositivos externos, el DSP hace uso del bus de datos principal como puerto paralelo.

- **Controlador DMA**. Este tipo de periférico permite efectuar transferencias de forma rápida, sin intervención del procesador. Esto es especialmente útil en aplicaciones en las que el volumen de datos a manejar es considerable; por ejemplo, procesamiento digital de imagen. En tal caso el DMA se utiliza para llevar bloques de datos desde una memoria externa de gran capacidad a la memoria interna, con objeto de que se puedan procesar a mayor velocidad.

- **Interfaz con un Host (HPI)**. En algunos casos los DSP funcionan como coprocesadores matemáticos, formando parte de un sistema global controlado por un procesador de propósito general. Para comunicarse con este procesador host, algunos DSP incorporan un puerto paralelo bidireccional de 8 o 16 bits, que podría incluso estar diseñado específicamente para comunicarse con un bus estándar, como el ISA o el PCI. Las comunicaciones a través de este puerto suelen realizarse mediante DMA de manera que sea posible transferir datos entre memoria y el puerto, sin intervención del procesador. El control del puerto se realiza mediante instrucciones específicas. En ocasiones esta interfaz permite incluso el control del funcionamiento del DSP en tareas tales como; forzar la ejecución de rutinas de tratamiento de interrupción, acceder a los registros internos del DSP o a su memoria, e incluso realizar la carga del código a ejecutar (*bootstraping*).

- **Puertos de Comunicación**. Se trata de puertos paralelo diseñados para comunicación entre los DSP del mismo tipo, y que pueden conectarse en red para implementar un sistema multiprocesador. Puesto que la anchura de estos puertos (8 bits) es menor que el tamaño de la palabra de datos de los DSP que los incluyen (32 bits), los puertos disponen de FIFOs para la fragmentación y re–ensamblado de los datos que se trasmiten a través de ellos. La comunicación a través de estos puertos suele estar asistida por DMA.

- **Puerto de I/O (GPIO)**. Estos grupos de pines en un DSP pueden habilitarse como entradas o salidas y se usan para propósitos de control, aunque también se pueden usar para transferencia de datos.

Los DSP diseñados para aplicaciones muy específicas, como el control de motores o sistemas de potencia, disponen de periféricos concretos para dichas aplicaciones, como pueden ser: generadores de señal PWM, temporizadores especiales para la implementación de tacómetros digitales, convertidores A/D, D/A, etc. En general se trata de DSP de baja escala, ya que el área utilizada del circuito integrado para estos periféricos limita el tamaño de los bancos de memoria interna y la sofisticación del CPU.

En la figura 4-12 se describe brevemente como se configura internamente un DSP, el cual contiene una gran variedad de periféricos: controlador de DMA, puertos serie y timers. En el diagrama se incluye la existencia de una arquitectura Harvard. Así aparecen múltiples bancos de memoria interna. Además, esta arquitectura se manifiesta también en el exterior del dispositivo con dos buses externos: bus principal y bus de expansión.

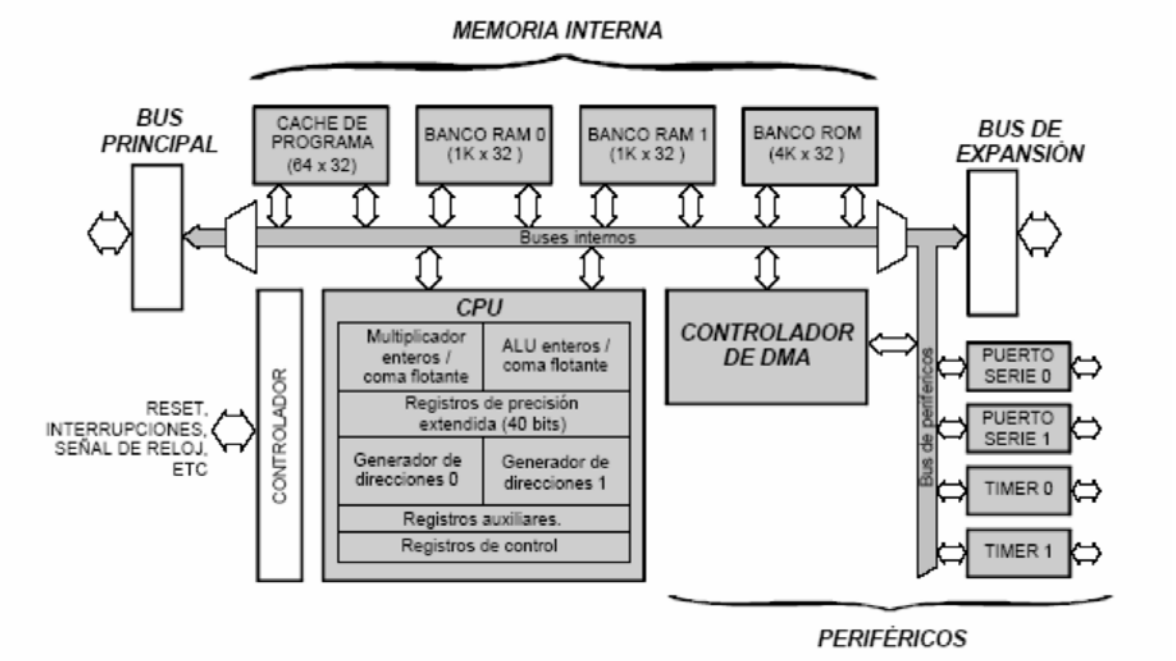


Figura 4-12 Arquitectura del TMS320C3x de Texas Instruments.

4.2.4. Criterios de Selección del DSP

En principio, todas las arquitecturas de un DSP parecen ser muy similares, pero no para un usuario que tiene bien definidas las necesidades y restricciones de una aplicación. Es decir, deben de valorarse las premisas que en algunos casos son muy particulares para determinar las características que debe tener el DSP. Por ejemplo, sí el objetivo es desarrollar un producto y se está limitado en el tiempo de lanzamiento al mercado, un DSP de punto flotante podría ser una buena elección, ya que su desarrollo de software es más simple que

los dispositivos de punto fijo. Sí el costo es la consideración primaria, existe un amplio número de DSPs de bajo costo, el problema es que vienen comercializados con limite de rendimiento y el software de desarrollo cambia continuamente. Sí el consumo de energía es una restricción clave, entonces la elección de un DSP de bajo voltaje con una arquitectura basada en un núcleo ASIC puede ser la solución para aplicaciones especiales.

En muchas aplicaciones, algunos de los factores mencionados son indispensables, sin embargo es importante identificar la arquitectura que mejor se adapte a las características de una determinada situación. Abajo se detallan las características básicas que debe reunir el DSP.

- **El tipo de aritmética utilizada y el ancho de palabra de datos**. La principal diferencia en el uso de DSP de punto fijo o punto flotante radica en cómo son los resultados de las operaciones de multiplicación manipuladas. Los procesadores de punto fijo tienen un hardware más simple que los de punto flotante, tomando en cuenta que éstos últimos son dispositivos más caros y de mayor consumo. Una recomendación para los diseñadores es que deben intentar el uso de un DSP con el menor ancho de palabra que su aplicación puede tolerar. En la figura 4-13 se observa la clasificación de los DSP por el tipo de aritmética que usan y la anchura de datos.

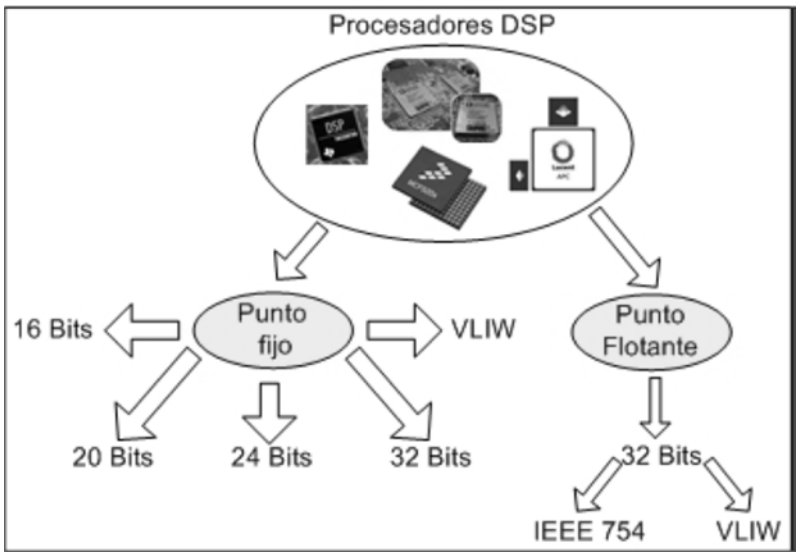


Figura 4-13 Clasificación de los DSP según su representación aritmética.

- **Velocidad**. Algunas arquitecturas claramente tienen ventaja en ciertas áreas. Una de esas áreas muy importante es la velocidad de ejecución ya

que los ciclos de tiempo de instrucción pueden variar significativamente en la ejecución de la aplicación. En general, las mediciones del rendimiento de un procesador deben tomarse con cierto escepticismo. En particular, las comparaciones basadas en MIPS y MFLOPS son muy inciertas, por la gran diferencia en el total de procesamiento que los procesadores pueden lograr con una sola instrucción u operación de punto flotante.

- **Tamaño de Memoria**. Una arquitectura de memoria eficiente es aquella que permite procesar las instrucciones y datos al ritmo que los demanda la CPU. Es preferible que los DSP dispongan de la mayor cantidad de memoria interna, ya que los accesos sobre ésta se realizan a mayor velocidad. La disponibilidad de memoria flash interna permite reducir la complejidad del sistema.

- **Periféricos integrados**. Esta consideración es una de las más importantes cuando se selecciona un DSP, por lo que se deben definir claramente las aplicaciones. La inclusión de periféricos o interfaz que le permitan comunicarse con el exterior en un mismo dispositivo puede tener un impacto significativo en el costo del procesador. Los DSP de punto fijo tienen a su disposición un conjunto de periféricos más variado que los de punto flotante, lo que los hace más eficientes en aplicaciones específicas. Por ejemplo la familia C2000 de Texas Instruments está enfocada a aplicaciones de control de motores, e incorporan a sus DSP periféricos tales como: *Timers*, PCI, DAC, ADC, GPIO, PWM, etc.

- **Consumo**. Este es un factor determinante en algunas aplicaciones, tales como las móviles, caso específico el de los teléfonos móviles. Actualmente se encuentran DSP con un voltaje nominal entre 3.0 y 3.3 V. Además, los fabricantes están agregando características de ahorro de energía controlado por medio de *software* o *hardware*. Estos modos de ahorro consisten en apagar ciertas secciones del procesador, así que el diseñador debe considerarlos si su aplicación así lo requiere.

- **Costo**. En algunas aplicaciones el DSP podría representar una pequeña fracción del costo total del sistema. En otras, el DSP podría comprometer un gran porcentaje del costo de inversión. En aplicaciones de gran consumo este aspecto puede prevalecer sobre otros que inciden más directamente sobre las prestaciones del DSP.

- **Rango Dinámico**. Es un parámetro que relaciona el tipo de aritmética utilizada y el ancho de la palabra de datos. Se define como la relación que existe entre el máximo y mínimo número que se pueden representar y por supuesto diferente de cero. A este respecto, los procesadores de punto flotante tienen un rango dinámico más amplio lo cual se traduce en una mejor precisión.

- **Soporte y *software* de desarrollo**. Cuando se selecciona un procesador, es importante considerar qué tipo de soporte proporciona el fabricante que ayude al proceso de diseño y problemas que puedan surgir. Tradicionalmente, el *software* para la programación de los DSP se escribe en lenguaje ensamblador, lenguaje "C", y debe ser extremadamente eficiente. Por ejemplo, Texas Instruments brinda en la siguiente dirección de Internet http://focus.ti.com/general/docs/dsnsuprt.tsp, servicios de información tales como: manuales técnicos, notas de aplicación, tutoriales, hojas de datos, ayuda en línea, etc. El software de desarrollo que suministra Texas Instrument para sus dispositivos es Code Composer Studio (CCS), y se suele utilizar código fuente en lenguaje ensamblador y C++, que permite usar una tarjeta de DSP en concreto.

4.3. Dispositivos lógicos programables

Históricamente, los PLA (*Programmable Logic Array*) fueron los primeros dispositivos lógicos programables (PLDs). Los PLA consistían en una matriz de puertas AND y otra de puertas OR con una estructura de dos niveles, con conexiones programables por el usuario. La estructura de los PLA sufrió mejoras y su coste disminuyó con la introducción de las PAL (*Programmable Array Logic*), en los que la matriz de puertas OR es fijo (no programable) y las GAL (*Generic Array Logic*) análogos a las PAL. Actualmente, tales

dispositivos se denominan de manera genérica como dispositivos lógicos programables (PLDs) y puede decirse que son los MSI de la industria de la lógica programable.

Estos dispositivos, antecesores de las FPGAs, ya se configuraban haciendo uso de un lenguaje de descripción de hardware. Sin embargo, la configuración de los primeros circuitos se realizaba, inicialmente, mediante la destrucción física de unos fusibles y sólo se podían programar una vez, con todas las limitaciones que esto suponía. Con el paso del tiempo, la tecnología permitió el borrado del dispositivo mediante exposición a rayos ultravioleta o bien eléctricamente y, por tanto, su reconfiguración. Los dispositivos lógicos programables evolucionaron con la idea de proporcionar al diseñador mayor flexibilidad y posibilidades de configuración, con lo que incorporaron registros, realimentaciones a los *arrays* programables, *buffers* tri-estado, etc.

4.3.1. Dispositivo lógico programable complejo (CPLD)

La siempre creciente capacidad de los circuitos integrados creó una oportunidad para los fabricantes de diseñar PLDs más grandes para aplicaciones más complejas de diseño digital. Sin embargo, un dispositivo demasiado grande usando una estructura cableada sería demasiado lento y, desde el punto de vista del fabricante, no se haría un uso efectivo en cuanto al costo del área del circuito integrado. Como conseguir esta estructura resultaba irrealizable, los fabricantes idearon un dispositivo lógico programable complejo (CPLD) que consistía, básicamente, en una colección de PLD individuales en un solo circuito integrado, junto a una estructura de interconexión programable que permitía que los PLD fueran conectados entre sí en el chip de la misma manera que un diseñador conectaría externamente PLDs individuales.

CPLD corresponde a las siglas de *Complex Programmable Logic Device*, es decir, Dispositivo Lógico Programable Complejo,con el que los fabricantes trataban de cumplir los siguientes objetivos:

- Aumentar las prestaciones de los PLD simples, integrando un número superior de puertas, y modificar la estructura de forma que se pudiese aprovechar lo mejor posible sus recursos (productos lógicos).

- Mantener los retardos de propagación de las PAL más rápidas.

- Permitir al diseñador predecir los retardos de propagación de las señales de salida, sin tener que esperar al *place and route* del PLD. De esta forma el diseñador puede detectar con antelación problemas de violaciones de tiempo, y en función de ello, hacer el diseño más correcto. Esta característica soluciona el problema de las FPGAs, referente a la imposibilidad de poder predecir sus retardos de propagación.

Estos objetivos determinaron que la estructura general de todos los CPLD fuese la de la figura 4-14:

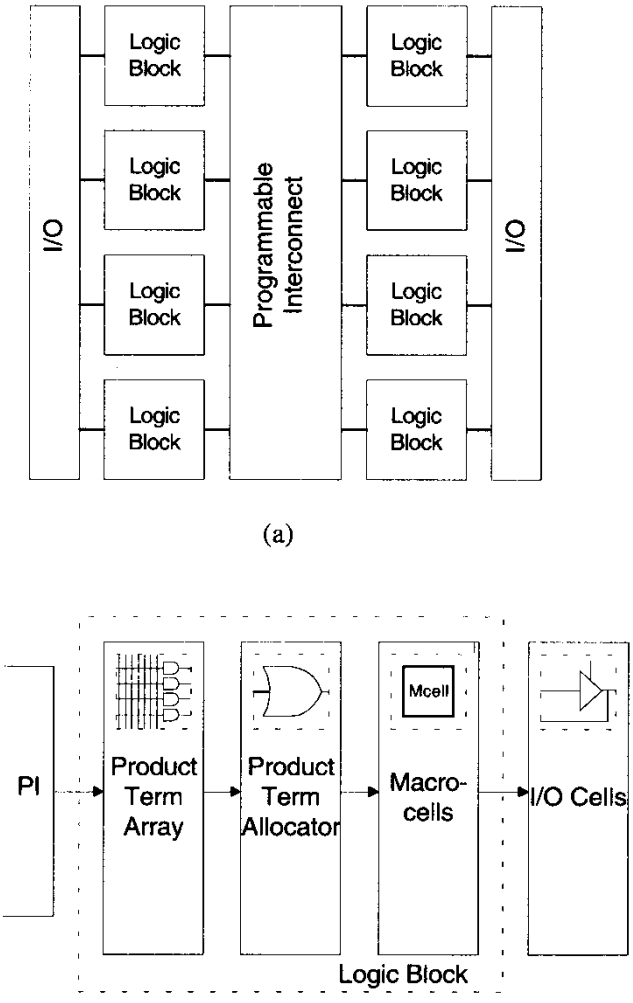


Figura 4-14 Arquitectura genérica de un CPLD y de un Bloque Lógico.

Los CPLD contienen múltiples bloques lógicos similares a un PLD simple, los cuales se comunican a través de una interconexión programable. Esta arquitectura utiliza más eficientemente el área de silicio que la de un PLD simple, por lo que mejora las prestaciones y reduce el coste.

Actualmente, las compañías Xilinx, Altera y Lattice fabrican CPLDs que difieren en las PLDs internas, tanto la matriz de puertas AND como en las macroceldas de salida, en los bloques de entrada y salida, y en la interconexión programable.

4.3.2. Matriz de puertas programable en campo (FPGA)

La invención del microprocesador retrasó al menos una década la aparición del primer dispositivo FPGA. Siempre teniendo en mente las ventajas e inconvenientes de los ASIC, Xilinx sacó al mercado un *Gate Array* programable por campo, es decir, la sustitución de la interconexión fija de los *Gate Arrays* por una serie de pistas metálicas conectables por transistores de paso controlados por un conjunto de bits de control almacenados en una memoria interna. Así, estos dispositivos surgen en 1985 con el nombre de LCA (*Logic Cell Array*), aunque posteriormente se renombraron como FPGA (*Field Programmable Gate Array*).

Básicamente, en una FPGA la lógica se divide en un gran número de bloques lógicos programables que suelen ser individualmente más pequeños que un PLD. Se encuentran distribuidos a través de todo el circuito integrado con nuna red de interconexiones programables y todo el *array* se encuentra rodeado de bloques de E/S programables (IOBs). Un bloque lógico programable (CLB o *slice*) de una FPGA es menos eficiente que un PLD típico, pero un chip FPGA contiene muchos más bloques lógicos que los PLD que contiene un CPLD del mismo tamaño.

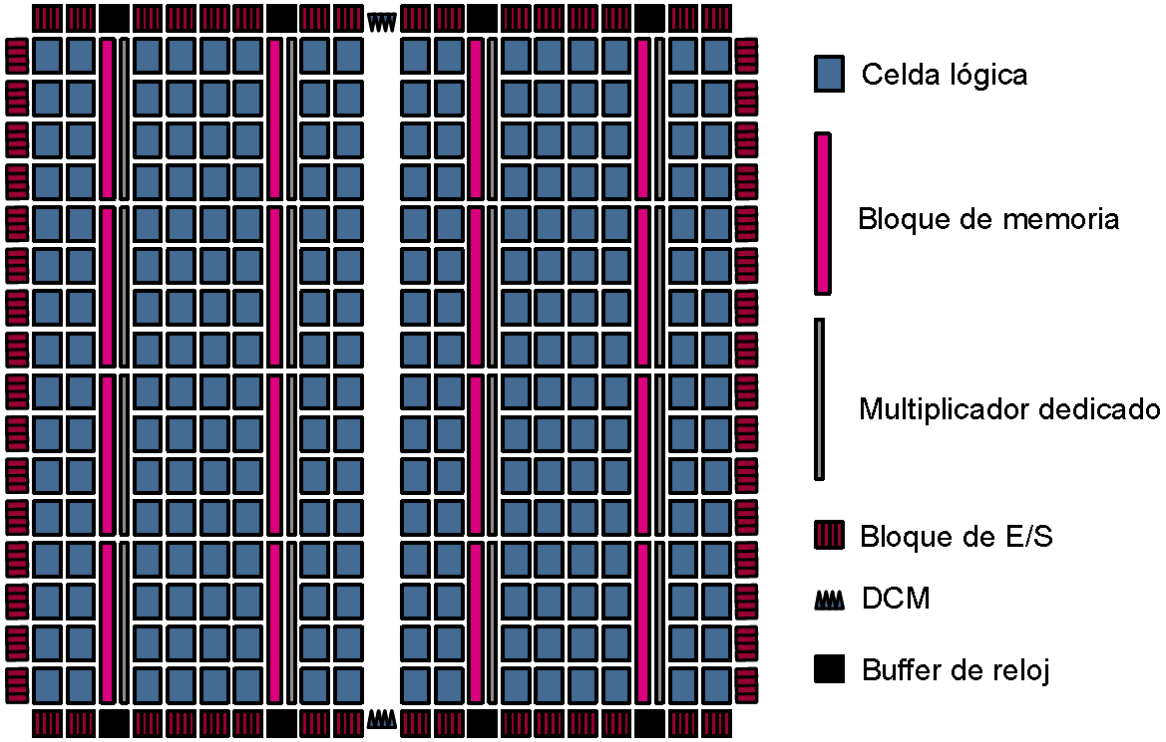


Figura 4-15 Arquitectura genérica de una FPGA.

Una FPGA (*Field Programmable Gate Array* - Matriz de Puertas Programable en Campo) es, al igual que un PLD y un CPLD, un ASIC programable, pero que requiere estrategias de programación diferentes y que, por tanto, su arquitectura no se basa en *arrays* tipo PLA. En otras palabras, un FPGA se compone de elementos con recursos no comprometidos que pueden ser seleccionados, configurados e interconectados por el usuario. Recordemos que en un PLD las interconexiones entre los elementos ya están hechas, solamente podemos habilitar o deshabilitar la interconexión; en el caso de un FPGA no hay nada interconectado.

Estos dispositivos se componen de cierto número de celdas lógicas configurables, que determinan la capacidad del dispositivo. Estas celdas son independientes entre sí y pueden interconectarse para formar un módulo más complejo. Dependiendo del fabricante, estos módulos pueden ser bloques configurables, como en las FPGA's de Xilinx, o bien, elementos de función fija formados por matrices de puertas, como en el caso de los dispositivos de Actel.

Los módulos en una FPGA, se interconectan por medio de canales de ruteo configurables. Al proceso de interconexión, se le conoce como enrutamiento y consiste en determinar la mejor estrategia de interconectar los módulos, ya sea en forma manual o mediante alguna herramienta de diseño electrónico. La gran ventaja de utilizar estos

dispositivos radica en que todo el desarrollo se lleva a cabo en un solo ambiente de trabajo. El diseñador propone la función lógica a realizar y en base a métodos de descripción define los parámetros de su problema. Esto se realiza por medio de código programable, que puede ser un lenguaje de descripción de hardware (VHDL o Verilog), o bien, un diagrama esquemático de conexiones.

Una vez delimitado el problema, se optimiza su representación lógica mediante métodos de minimización (síntesis lógica); posteriormente se simula, lógica y eléctricamente (simulación). Con la ayuda del software actual, que es muy sofisticado, este último paso puede llegar a ser muy aproximado al comportamiento real del dispositivo. Se selecciona, entonces, el dispositivo que mejor se adapte a las condiciones de nuestro problema según criterios de capacidad, velocidad, consumo de energía, costo, etc., y finalmente se programa en campo. Después de la programación se puede ejercer una optimización y si se cree necesario hacer modificaciones en el diseño. Sólo bastaría con hacerlas en el código y repetir los pasos anteriormente descritos. El dispositivo se podrá reconfigurar nuevamente hasta en unos 200 intentos, que según el fabricante, es el tiempo de vida promedio.

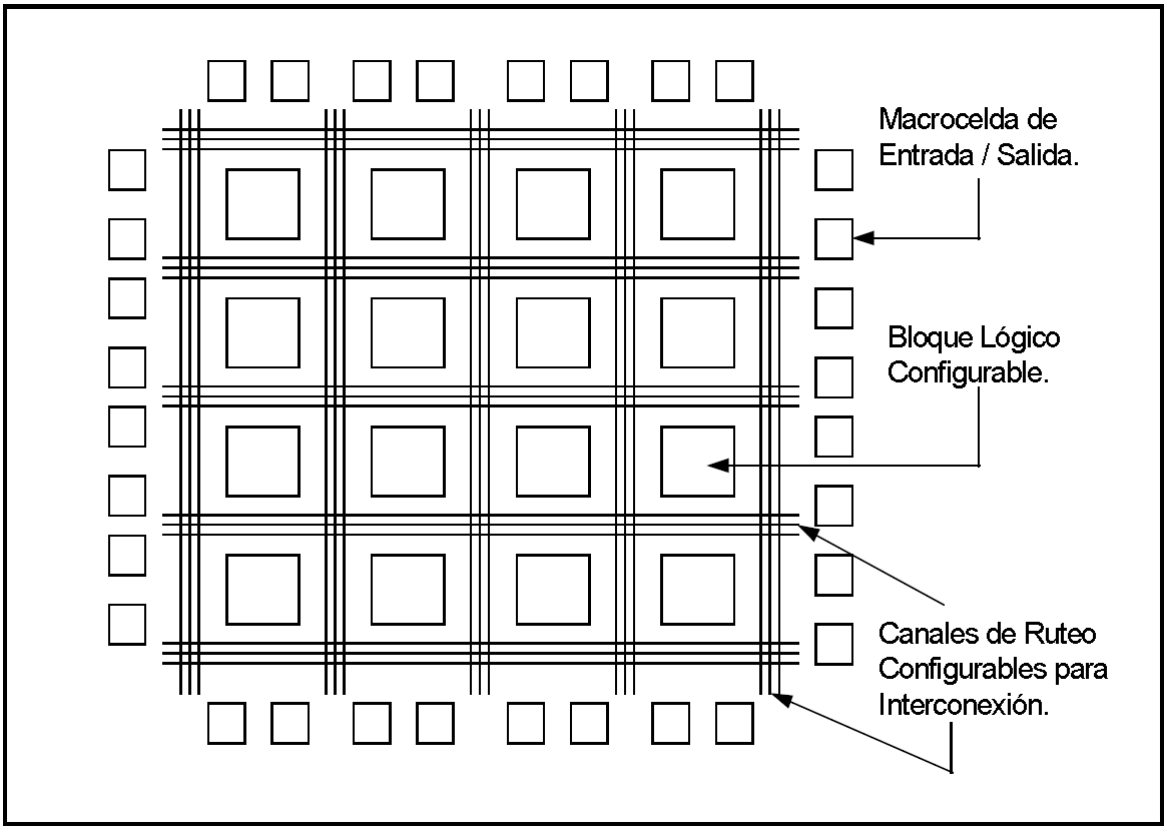


Figura 4-16 Arquitectura demostrativa de una FPGA de Xilinx.

Las principales ventajas de diseñar sobre FPGA's las enumero a continuación:

- Minimización del número de componentes en un diseño. Con esto se reducen los gastos de inventario, inspección y prueba, así como el número de fallas a nivel de circuito impreso, propiciando un ahorro de espacio físico.

- Reducción en el tiempo de diseño. Debido a su naturaleza programable, reducen el tiempo y los costos de desarrollo, no sólo de nuevos productos sino también de aquellos que requieren modificaciones, ya que son reutilizables tantas veces como sea necesario. Esto último se debe a que los cambios en el diseño son realizados mediante una nueva programación que se prueba inmediatamente si se está utilizando un dispositivo programable en el mismo circuito (*In System*).

- Uso de una gran variedad de herramientas de Diseño Asistido por Computador (CAD), disponibles actualmente en el mercado. Estas herramientas promueven y facilitan el diseño sobre este tipo de dispositivos. Así mismo, no se requiere de grandes recursos de cómputo.

4.4. Familias de FPGAs de Xilinx

Los diseñadores coinciden en afirmar que desde el punto de vista usuario, existen tres fabricantes mayoritarios en la distribución de FPGAs y software de soporte: Xilinx, Altera y Actel. En el mercado mundial podemos encontrar otros tantos con producciones menores pero que figuran también como FPGAs útiles: Lucent, Texas Instruments, Philips, QuickLogic, Cypress, Atmel, etc.

Xilinx está considerado como uno de los fabricantes más fuertes a nivel mundial. Sus FPGAs (también fabrica PLDs y CPLDs) están basados en la tecnología SRAM y son dispositivos programables múltiples veces, programables en sistema (*In System*).

En la tabla 4-1 describo la evolución y características generales de las distintas familias de FPGAs de Xilinx:

Tabla 4-1 Evolución y características de las distintas familias de FPGAs de Xilinx

Familia	Año de creación	Número máximo de puertas	Novedades
XC2000	1985	1800	Configuración SRAM

XC3000	1987	7500	Mayor número de puertas
XC4000/Spartan	1991	180000	Memoria RAM
Virtex/Spartan II	1998	3M2/600K	BRAMs, DLLs
Virtex II	2000	8M	DCMs, multiplicadores
Spartan 3	2003	5M	Bajo coste
Virtex II-Pro	2002	4M	Power PC, Rocket I/O
Virtex 4	2005	8M	Ethernet MACs, DSP *slice*
Virtex 5	2006	11M	DSP Slice avanzado, BRAM 32KBits

La estructura de estos dispositivos está compuesta por módulos lógicos llamados por Xilinx, CLBs (*Configurable Logic Blocks* – Bloques Lógicos Configurables), basados en Tablas de Búsqueda (*LookUp Tables* - LUTs). Cada CLB contiene circuitos que les permiten realizar operaciones aritméticas eficientes. También los usuarios pueden configurar las tablas de búsqueda como celdas *read/write* (lectura/escritura) de RAM. A la vez, a partir de la serie XC 4000, se incluye un generador interno de señal de reloj con 5 diferentes frecuencias. Además de los CLBs, los FPGA de este fabricante incluyen otros bloques complejos que configuran la entrada de los pines físicos que conectan el interior del dispositivo con el exterior, a estos bloques se les llama IOBs (*Input/Output Blocks* – Bloques de Entrada/Salida). Cada IOB contiene una lógica compleja que permite que un pin pueda actuar como entrada, salida o un tercer estado.

La serie XC Virtex, o tan solo Virtex, es la más nueva de todas las propias de Xilinx. Se dice que los dispositivos pertenecientes a esta familia son los más rápidos (velocidades de trabajo de hasta 250 Mhz), densos en compuertas y menor consumo de potencia, pero por la misma razón son los más costosos.

La serie Spartan surgió como una opción para sustituir diseños probados de menos de 15.000 puertas por dispositivos de bajo costo y alto desarrollo (además incluyen el soporte de los *cores* prediseñados), pero sacrificando algunas características que manejan las series estándar de Xilinx, como la XC4000 tradicional o la serie Virtex. Las series estándar XC3000 y su sucesora, la 4000, son series que muestran la mayor parte de las características funcionales de los FPGAs de Xilinx, pero con la gran desventaja que son dispositivos de baja densidad de compuertas. Sin embargo, el uso de la serie XC4000 (en especial el XC40003E) es muy favorecido para el diseño e implementación de prototipos de bajo impacto.

Las FPGAs que utilizadas en esta tesis son de la serie Spartan de bajo coste, en concreto la Spartan 3 y la Spartan 6.

4.4.1. Arquitectura de la FPGA Spartan 3 de Xilinx

Las FPGA Spartan 3 de Xilinx [41] están conformadas por un conjunto de Bloques Lógicos Configurables (*Configurable Logic Blocks*: CLBs) rodeados por un perímetro de Bloques Programables de entrada/salida (*Programmable Input/Output Blocks*: IOBs). Estos elementos funcionales están interconectados por una jerarquía de canales de conexión (*Routing Channels*), la que incluye una red de baja capacitancia para la distribución de señales de reloj de alta frecuencia. Adicionalmente el dispositivo dispone de bloques de memoria RAM de doble puerto, cuyos anchos de buses son configurables, y con varios bloques de multiplicadores dedicados de 18 X 18 bits.

Los elementos funcionales programables que la componen son los siguientes:

- **Bloques de entrada/salida (*Input/Output Blocks* – IOBs):** Controlan el flujo de datos entre los pines de entrada/salida y la lógica interna del dispositivo. Soportan flujo bidireccional más operación tri–estado y un conjunto de estándares de voltaje e impedancia controlados de manera digital.

- **Bloques Lógicos configurables (*Configurable Logic Blocks* – CLBs):** Contienen *Look-Up Tables* basadas en tecnología RAM (LUTs) para implementar funciones lógicas y elementos de almacenamiento que pueden ser usados como *flip-flops* o como *latches*.

- **Bloques de memoria RAM (*Block* RAM):** Proveen almacenamiento de datos en bloques de 18 Kbits con dos puertos independientes cada uno.

- **Bloques de multiplicación** que aceptan dos números binarios de 18 bit como entrada y entregan uno de 36 bits.

- **Administradores digitales de reloj (*Digital Clock Managers – DCMs*):** Estos elementos proveen funciones digitales auto calibradas, las que se encargan de distribuir, realizar el trazado arbitrariamente en pocos grados, desfasar en 90, 180, y 270 grados, dividir y multiplicar las señales de reloj de todo el circuito.

Los elementos descritos están organizados como se muestra en la figura 4-17. Un anillo de IOBs rodea un *array* regular de CLBs. Atraviesa este *array* una columna de bloques de memoria RAM, compuesta por varios bloques de 18 Kbit, cada uno de los cuales está asociado con un multiplicador dedicado. Los DCMs están colocados en los extremos de dichas columnas.

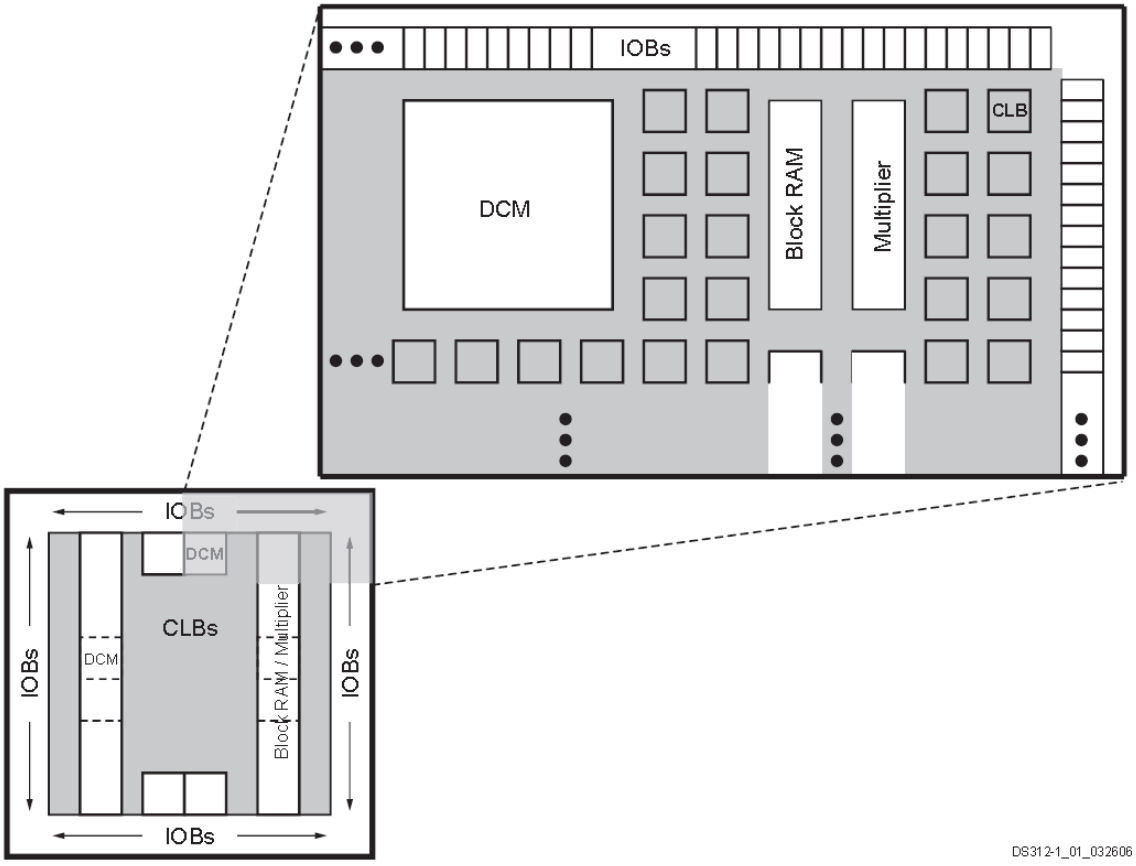


Figura 4-17 Arquitectura de la Spartan 3 de Xilinx.

A continuación realizo una descripción más detallada de cada uno de los elementos funcionales de esta FPGA.

4.4.1.1. Bloques de entrada/salida (IOB)

Los bloques de Entrada/Salida de las FPGAs cumplen la misma función que las macroceldas de salida en otros dispositivos lógicos programables, pero con más controles lógicos, entre los que se incluyen, configuraciones de entrada y salida combinacionales o registradas, alta impedancia, elementos de retardo, controles analógicos y otros. Suministran una interfaz bidireccional programable entre un pin de entrada/salida y la lógica interna de la FPGA. Un diagrama simplificado de la estructura interna de un IOB aparece en la figura 4-18.

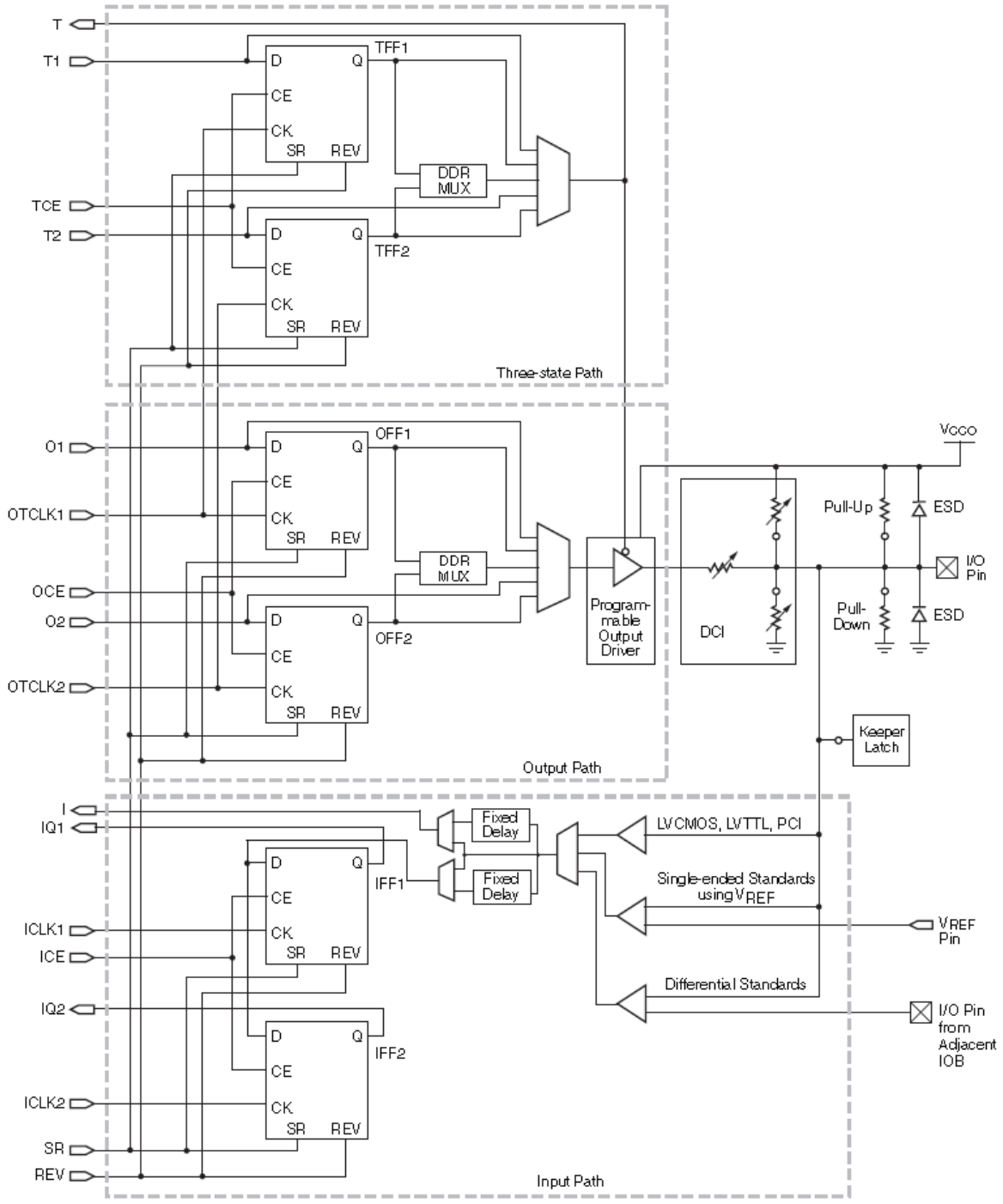


Figura 4-18 Diagrama simplificado de un IOB de la Spartan 3.

Hay tres rutas para señales en un IOB: la ruta de salida, la ruta de entrada y la ruta tri-estado. Cada ruta tiene su propio par de elementos de almacenamiento que pueden actuar tanto como registros o como *latches*. Las tres rutas principales se describen a continuación:

- La ruta de entrada, que lleva datos desde el *pad*, que está unido al pin del *package*, a través de un elemento de retardo opcional programable, directamente a la línea I. A continuación del elemento de retardo hay

rutas alternativas a través de un par de elementos de almacenamiento hacia las líneas IQ1 e IQ2. Las tres salidas del IOB todas conducen a la lógica interna de la FPGA.

- La ruta de salida, que parte con las líneas O1 y O2, lleva datos desde la lógica interna de la FPGA, a través de un multiplexor y del *driver* tri–estado hacia el *pad* del IOB. En suma a esta ruta directa, el multiplexor da la opción de insertar un par de elementos de almacenamiento.

- La ruta tri–estado determina cuándo el *driver* de salida está en alta impedancia. Las líneas T1 y T2 llevan datos desde la lógica interna a través de un multiplexor hacia el driver de salida. En suma a esta ruta directa, el multiplexor da la opción de entregar un par de elementos de almacenamiento.

Todas las rutas de señales que entran al IOB, incluidas aquellas asociadas con los elementos de almacenamiento tienen una opción de inversión. Cualquier inversor colocado (en la programación) en estas rutas es automáticamente absorbido dentro del IOB.

Hay tres pares de elementos de almacenamiento en cada IOB, un par para cada uno de las tres rutas. Es posible configurar cada uno de esos elementos como un flip-flop D activo por flanco o como un *latch* activo por nivel. Estos elementos son controlados con la misma red de distribución de relojes que se utiliza para todo el sistema.

El par de elementos de almacenamiento tanto de la ruta de salida o de la del *driver* tri–estado pueden ser usados en conjunto, con un multiplexor especial para producir transmisión de doble tasa de datos (DDR). Esto se logra tomando datos sincronizados con el flanco de subida del reloj y convirtiéndolos en bits sincronizados tanto con el flanco de subida como con el de bajada. A esta combinación de dos registros y un multiplexor se le llama flip flop tipo D de doble tasa de datos (FDDR).

Cada IOB cuenta además con otros elementos, entre los cuales cuentan las resistencias de *Pull-Up* y de *Pull-Down*, que tienen el objetivo de establecer niveles altos o bajos respectivamente en las salidas de los IOBs que no están en uso; un circuito de retención del último nivel lógico que se mantiene, después de que todos los *drivers* hayan sido apagados, lo que es útil para cuidar que las líneas de un bus no “floten”, cuando los drivers

conectados están en alta impedancia; un circuito de protección para descargas electro–estáticas (protección ESD), que utiliza diodos de protección.

Finalmente cada IOB cuenta con un control para el *slew rate* y para la corriente de salida máxima. El primero otorga la posibilidad de elegir una tasa alta de cambio de nivel (con bajo *slew rate*) o una tasa máxima menor, pero con un control de transiciones, para la utilización de los puertos en la integración a buses, donde al pasar de alta impedancia a un nivel de voltaje suele producirse transiciones inesperadas. El segundo entrega siete niveles diferentes de corrientes máximas tanto para el estándar CMOS como para el TTL, lo que permite adaptarse a dispositivos que necesitan mayores corrientes para su activación; en el caso del estándar LVCMOS a 2.5V el rango de corrientes es de 2 a 24 mA.(2, 4, 6, 8, 12, 16, 24 mA).

Los IOB soportan 17 estándares de señales de salida de terminación única y seis de señal diferencial; también cuentan con un sistema integrado, para coincidir con la impedancia de las líneas de transmisión que llegan a la FPGA, llamado Control Digital de Impedancia (DCI), el que permite elegir hasta 5 tipos diferentes de terminaciones, utilizando una red de resistencias internas que se ajustan en serie o en paralelo, dependiendo de las necesidades del estándar elegido.

4.4.1.2. Bloques de lógica configurable (CLB)

En los *slices* se realiza la mayor parte de la funcionalidad de la FPGA y suelen estar agrupados de 2 en 2 o de 4 en 4 formando bloques lógicos configurables (CLBs). Dentro de este componente encontramos los módulos LUT, registros y multiplexores programables en un número que depende de la familia de FPGA. Los elementos programables más importantes son los generadores reprogramables de función lógica, realizadas por las denominadas LUT (*Look–Up Table*) o tablas de búsqueda, que son celdas de memoria SRAM y multiplexores para seleccionar la salida. Los generadores de función pueden diseñarse para cualquier número de variables que se desee sin más que aumentar el tamaño de la memoria SRAM y la ubicación de selectores que escojan un solo valor almacenado para cada combinación de valores de las variables.

Sin embargo, el número de variables con que pueden diseñarse las LUTs no es trivial. Si intentamos realizar una LUT con un número de entradas elevado, el área que ocuparía sería relativamente grande, con lo que el número de *slices* dentro de la FPGA se reduciría.

Si, por el contrario, se opta por LUTs con pocas entradas, cabrían muchos *slices*, pero el ruteo sería complicado, se necesitarían muchas conexiones por lo que el retardo debido al cableado entre los *slices* sería importante.

Cualquier función lógica que se desee implementar con un número de variables mayor se realiza usando varias LUTs, ya que dicha función siempre podrá ponerse en función de varias funciones de 4 variables aplicando el conocido teorema de *Shannon* las veces que sea necesario.

$$F(X_1, X_2, \ldots . X_n) = X_1 \cdot F(1, X_2, \ldots X_n) + \overline{X_1} \cdot F\ (0, X_2, \ldots X_n)$$

En general, los *slices* contienen alguna lógica adicional aparte de las LUTs para aumentar las prestaciones y la eficiencia de estos bloques, como biestables para obtener salidas registradas o lógica para implementar eficientemente comparadores, contadores o sumadores serie. Además, como las funciones lógicas se generan en realidad a través de memorias SRAM, los propios *slices* se pueden configurar para usarlos como bloques de memoria en lugar de lógica, es lo que se denomina memoria distribuida para diferenciarla de bloques de memoria específicos que pudiera haber en la FPGA. Los *slices* más próximos suelen agruparse siguiendo esta filosofía en grupos denominados bloques lógicos configurables (CLB).

En la Spartan 3 cada CLB está compuesto de cuatro *slices* interconectados entre sí, tal como se muestra en la figura 4-19. Las cuatro *slices* que componen un CLB tienen los siguientes elementos en común: dos generadores de funciones lógicas, dos elementos de almacenamiento, multiplexores de función amplia, lógica de acarreo y puertas aritméticas, tal como se muestra en la figura 4-20. Los dos pares de *slices* usan estos elementos para implementar funciones lógicas y aritméticas de ROM. Además de lo anterior, el par de la izquierda soporta dos funciones adicionales: almacenamiento de datos usando RAM distribuida y desplazamiento de datos con registros de 16 bits.

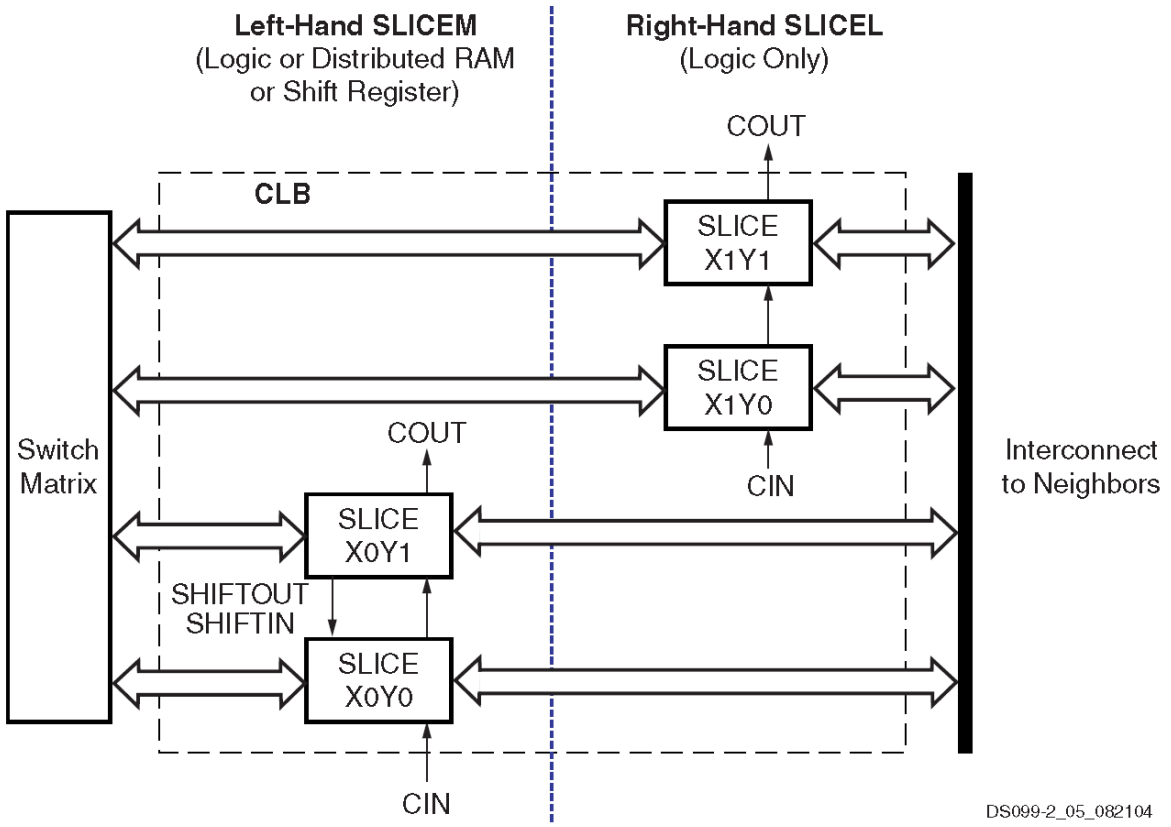


Figura 4-19 *Array* de slices en un CLB de la Spartan 3.

El generador de funciones basado en RAM (LUT) es el recurso principal para implementar funciones lógicas dentro de la FPGA. Más aún, las LUTs en cada par de *slices* del lado izquierdo pueden ser configuradas como RAM distribuida o como un registro de desplazamiento de 16 bits. Los generadores de funciones ubicados en las porciones superiores e inferiores del *slice* son referidos como “G-LUT” y “F-LUT” respectivamente en la figura 4-20.

El elemento de almacenamiento, el cual es programable tanto como un flip flop tipo D o como un *latch* sensible a nivel, provee un medio para sincronizar datos a una señal de reloj, entre otros usos. Estos elementos de almacenamiento, que se encuentran en las porciones superiores e inferiores de la slice son llamados “FFY” y “FFX”, respectivamente.

Los multiplexores de función amplia combinan las LUTs para permitir operaciones lógicas más complejas, cada *slice* tiene dos de éstos, en la figura 4-20 corresponden a F5MUX y F1MUX.

La cadena de acarreo, en combinación con varias puertas lógicas dedicadas, soportan implementaciones rápidas de operaciones matemáticas. La cadena de acarreo entra en el

slice como CIN y sale como COUT. Cinco multiplexores controlan la cadena: CYINIT, CY0F y CYMUXF en la parte inferior, así como CY0G y CYMUXG en la parte superior. La lógica aritmética dedicada incluye puertas XOR y AND en cada parte del slice.

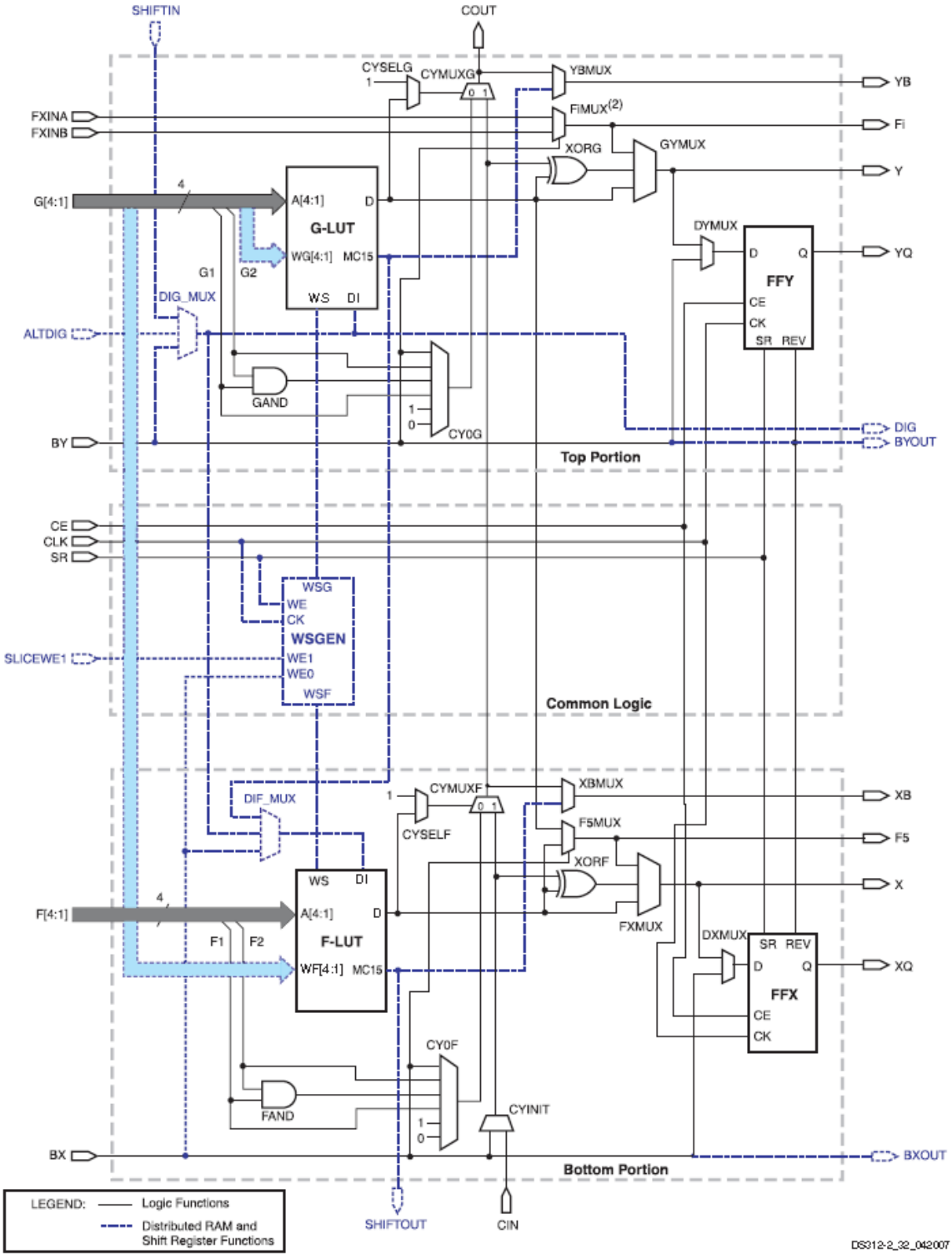


Figura 4-20 Diagrama simplificado de una slice del lado izquierdo de un CLB.

Con un rol central en la operación de cada *slice* se encuentran dos rutas de datos casi idénticas. Para la descripción que prosigue se usan los nombres de la parte inferior de la

figura 4-20. La ruta básica tiene su origen en la matriz de *switches* de interconexión colocada fuera del CLB. Cuatro líneas, F1 a F4 entran en el *slice* y se conectan directamente a la LUT. Una vez dentro del *slice*, la ruta de los 4 bits inferiores pasa a través de un generador de funciones F que realiza operaciones lógicas. La ruta de salida del generador de funciones, D, ofrece cinco posibles rutas posibles:

- Salir del *slice* por la línea X y volver a interconectarse.
- Dentro del *slice*, X sirve como entrada al DXMUX que alimenta la entrada de datos D, correspondiente al elemento de almacenamiento FFY. La salida Q de este elemento maneja la ruta XQ que sale del *slice*.
- Controlar el multiplexor CYMUXF de la cadena de acarreo.
- Con la cadena de acarreo, servir como una entrada a la puerta XORF, que realiza operaciones aritméticas y produce el resultado en X.
- Controlar el multiplexor F5MUX para implementar funciones lógicas mayores de 4 bits. Las salidas D de los F-LUT y G-LUT sirven de entradas de datos para este multiplexor.

En resumen a los caminos lógicos principales descritos anteriormente, existen dos rutas de *bypass* que entran al *slice* como BX y BY. Una vez dentro de la FPGA, BX en la parte inferior del *slice* (o BY en la parte superior) puede tomar cualquiera de varias ramas diferentes:

- Hacer *bypass* de la LUT y del elemento de almacenamiento, para salir del *slice* como BXOUT y volver a interconectarse.
- Hacer *bypass* a la LUT, y luego pasar a través del elemento de almacenamiento, para posteriormente salir como XQ.
- Controlar el multiplexor F5MUX.
- Servir como una entrada a la cadena de acarreo a través de los multiplexores.
- Manejar la entrada DI de la LUT.

- BY puede controlar la entrada REV de FFY y de FFX.

- Finalmente, el multiplexor DIG_MUX puede derivar la ruta BY hacia la línea DIG que sale del *slice*.

Cada una de las dos LUTs (F y G) de un *slice* tiene cuatro entradas lógicas (A1–A4) y una única salida D. Esto permite programar cualquier operación lógica booleana de cuatro variables en este dispositivo. Además, los multiplexores de función amplia pueden usarse para combinar LUTs dentro del mismo CLB o incluso a través de diferentes CLBs, haciendo posible funciones con mayor número de variables.

Las LUT de ambos pares de *slices* dentro de un CLB no sólo soportan las funciones descritas, sino que también pueden funcionar como una ROM con datos inicializados al momento de configurar la FPGA. Las LUTs del lado izquierdo de cada CLB soportan además dos funciones adicionales: primero, es posible programarlas como RAM distribuida, lo que permite contar con espacios de memoria de 16 bits en cualquier parte de la topología de la FPGA. Segundo, es posible programar una de estas LUTs como un registro de desplazamiento de 16 bits, con lo que se pueden producir retardos de hasta 16 bits o combinaciones de varias LUTs pueden producirlos de cualquier tamaño de bits.

4.4.1.3. Bloques de memoria RAM (Block RAM)

La Spartan 3 tiene 24 bloques de 18 Kbits de memoria RAM. El ancho del bus de datos frente al de direcciones (relación de aspecto) de cada bloque es configurable y se puede combinar varios de éstos para formar memorias de mayor longitud de palabra o de mayor tamaño.

Tal como se muestra en la figura 4-21, los bloques de RAM tienen una estructura de doble puerto. Dos puertos idénticos llamados A y B permiten acceso independiente al mismo rango de memoria, que tiene una capacidad máxima de 18.432 bits – o 16.384 cuando no se usan las líneas de paridad. Cada puerto tiene su propio conjunto de líneas de control, de datos y de reloj para las operaciones síncronas de lectura y escritura. Estas operaciones tienen lugar de manera totalmente independiente en cada uno de los puertos.

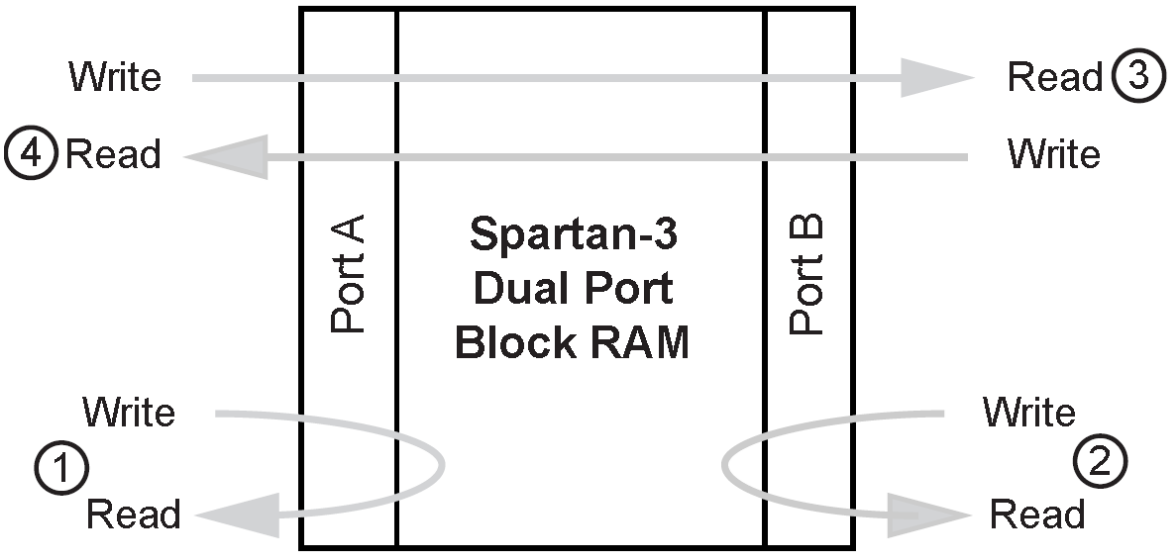


Figura 4-21 Diagrama de un bloque de RAM dedicado de la Spartan 3.

4.4.1.4. Bloques de multiplicación empotrados

La Spartan 3 provee multiplicadores empotrados que aceptan palabras de 18 bits como entrada y entregan productos de 36 bits. Los buses de entrada de estos multiplicadores aceptan datos en complemento dos (tanto 18 bits con signo, como 17 bits sin signo). Para cada bloque de RAM hay un multiplicador inmediatamente colocado y conectado; dicha proximidad permite un manejo eficiente de los datos.

A estos bloques (MULT18x18SIO) les dedicaré más adelante un apartado por ser los componentes principales de la operación MAC y, por tanto, de la convolución.

4.4.1.5. Administradores digitales de reloj (Digital Clock Managers–DCM)

La Spartan 3 dispone de 4 bloques para el control de todos los aspectos relacionados con la frecuencia, la fase y el *skew* de la red de relojes de la FPGA. Cada DCM tiene cuatro componentes funcionales: el *Delay-Locked Loop* (DLL), el sintetizador digital de frecuencia (DFS) y el desplazador de fase (PS), además de incluir cierta lógica para el estado. La figura 4-22 muestra un diagrama de bloques de este elemento funcional de la FPGA.

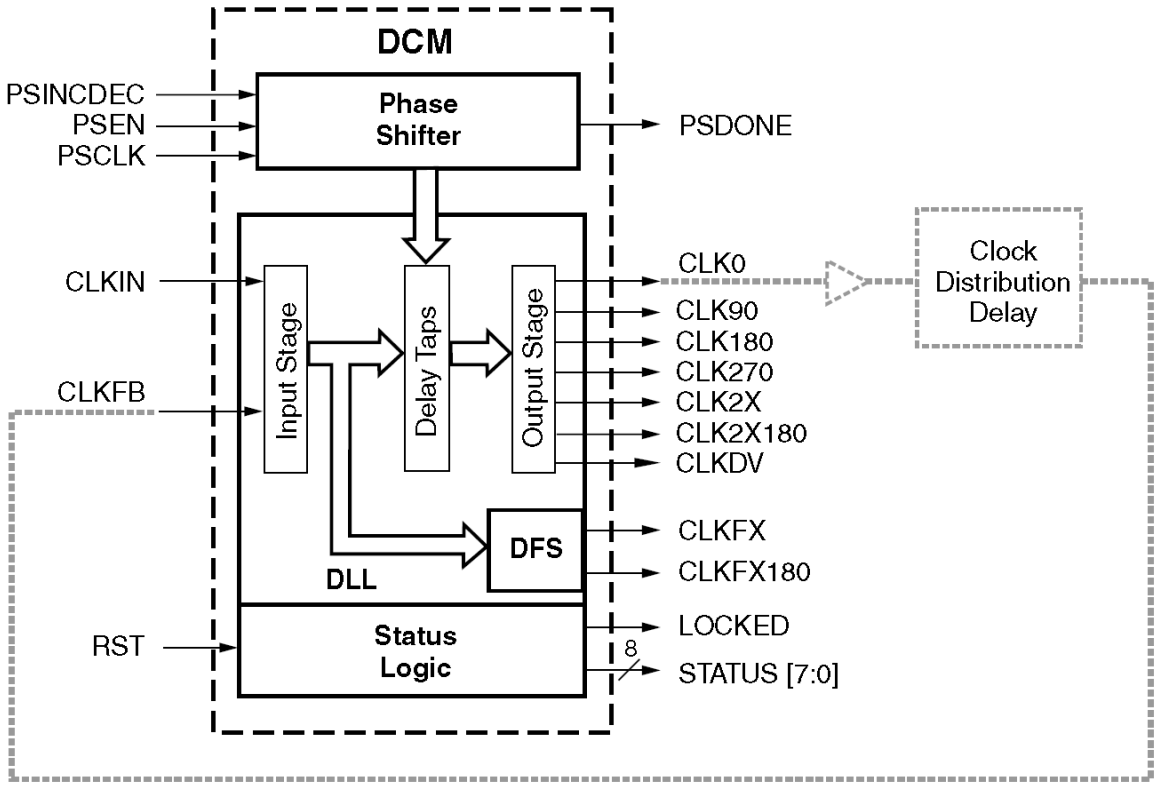


Figura 4-22 Diagrama de un bloque DCM de la Spartan 3.

El DCM realiza tres funciones principales:

- Eliminación de *skew* de reloj: El concepto de *skew* describe el grado al cual las señales de reloj pueden, bajo circunstancias normales, desviarse del alineamiento de la fase cero. Ello ocurre cuando pequeñas diferencias en los retardos de las rutas causan que la señal de reloj llegue a diferentes puntos del circuito en tiempos diferentes. Este *skew* de reloj puede incrementar los requerimientos de *set-up time* y de *hold time*, lo que puede perjudicar el desarrollo de aplicaciones de alta frecuencia. El DCM elimina el *skew* de reloj alineando la salida de la señal de reloj que genera con otra versión de la misma señal que es retroalimentada. Como resultado se establece una relación de desfase cero entre ambas señales.

- Síntesis de frecuencia: Provisto de una señal de reloj de entrada, el DCM puede generar diferentes relojes de salida. Ello se logra multiplicando y/o dividiendo la frecuencia del reloj de entrada.

- Desplazamiento de fase: El DCM puede producir desfases controlados de la señal de reloj de entrada y producir con ello relojes de salida con diferentes fases.

El DLL tiene como principal función eliminar el *skew* de reloj. La ruta principal del DLL consiste en una etapa de entrada, seguida por una serie de elementos de retardo discreto o *taps*, los cuales conducen a una etapa de salida. Esta ruta, junto con lógica para detección de fase y control conforman un sistema completo con retroalimentación, tal como se muestra en la figura 4-23.

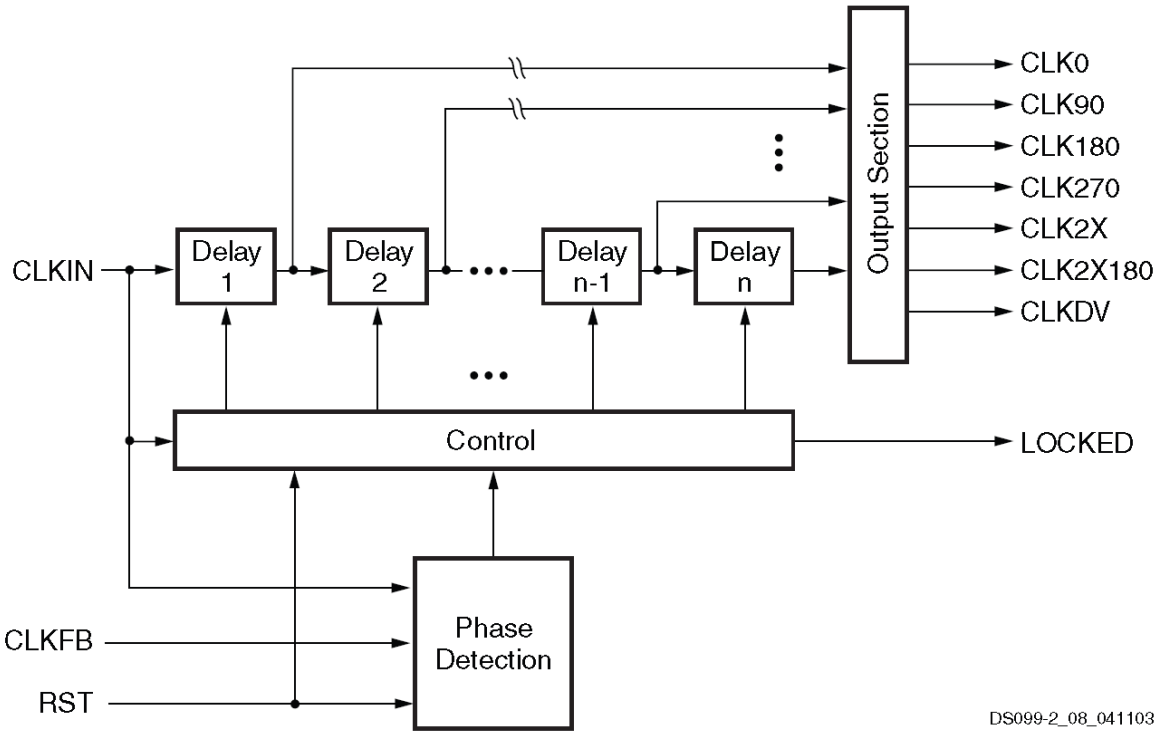


Figura 4-23 Diagrama funcional simplificado del *Delay-Locked Loop* (DLL).

La señal CLK0 es entregada a la red de distribución de señales de reloj de la FPGA, que sincroniza todo los registros del circuito que ha sido configurado. Estos registros pueden ser tanto internos como externos a la FPGA. Después de pasar por dicha red, la señal de reloj retorna al DLL a través de la entrada CLKFB. El bloque de control del DLL mide el error de fase entre ambas señales, que es una medida del *skew* de reloj que toda la red introduce. El bloque de control activa el número apropiado de elementos de retardo para cancelar el *skew* de reloj. Una vez que se ha eliminado el desfase, se eleva la señal LOCKED, que indica la puesta en fase del reloj con respecto a la retroalimentación.

Las señales de reloj tienen una red dedicada especial para su distribución. Esta red tiene ocho entradas globales, por medio de *buffers*. La red tiene baja capacitancia y produce muy bajo *skew* de reloj, lo que la hace adecuada para conducir señales de alta frecuencia. Tal como se muestra en la figura 4-24, las entradas GCLK0 a GCLK3 están situadas en la parte inferior de la FPGA, mientras que las entradas GCLK4 a GCLK7 están colocadas en la parte superior. Se puede conducir cualquiera de dichas entradas hacia cada uno de los CLBs, por medio de las líneas principales, que se observan en negro grueso en la figura 4-24. Las líneas más delgadas representan líneas que conducen hacia los elementos síncronos de cada una de los *slices* de los CLBs.

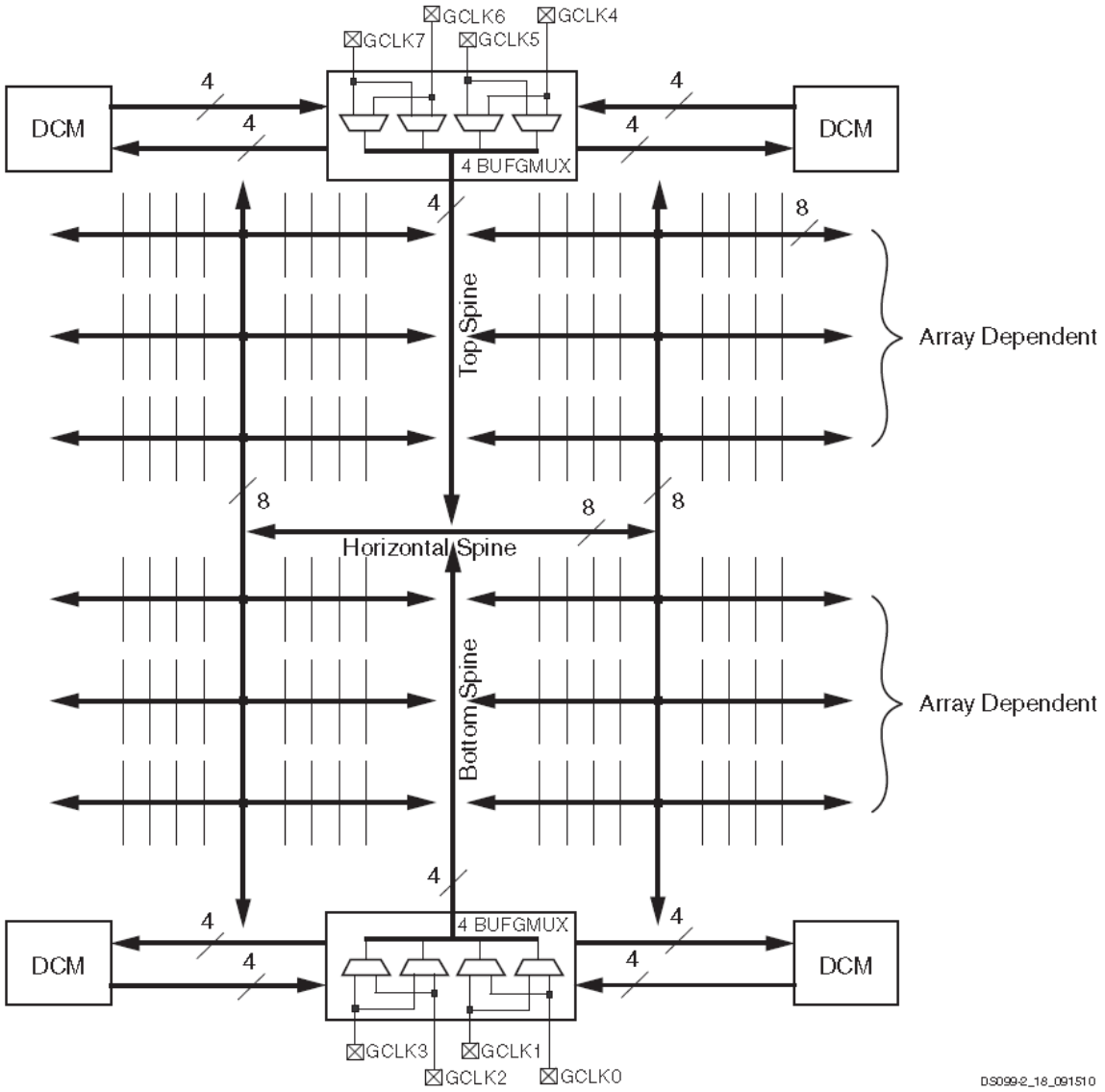


Figura 4-24 Red de distribución de señales de reloj de la Spartan 3.

4.4.1.6. Interconexión programable de la FPGA

Cada CLB en la FPGA se encuentra incrustado en la estructura de interconexión, que están compuestos de cables con conexiones programables para ellos. Inicialmente, se disponía de unas interconexiones heterogéneas de propósito general, aunque en la década de los 90 se evolucionó hacia una estructura de interconexión jerárquica. Las líneas del grupo del reloj están optimizadas para su uso como entradas de reloj a los CLB, proporcionando un retardo corto. El conjunto de líneas simples se optimizan para conectividad flexible entre bloques adyacentes, pero en mayor cantidad y sin la limitación unidireccional de las líneas directas.

Sería posible conectar dos CLBs no adyacentes usando líneas simples, pero deberían pasar por un conmutador programable para cada salto, lo que agregaría retardos adicionales. Las líneas de los grupos doble viajan pasados dos CLBs antes de llegar a un conmutador, de modo que proporcionan retardos más cortos para conexiones más largas. Para conexiones muy largas, se emplean los grupos largos, que no pasan por ningún conmutador programable y recorren toda la FPGA en vertical u horizontal.

La red de interconexión conduce las señales entre varios elementos funcionales de la Spartan 3. Hay cuatro tipos de interconexiones: *Long lines*, *Hex lines*, *Double lines* y *Direct lines*. *Long lines* son aquellas que conectan una salida de cada seis CLBs (figura 4-25a). Debido a su baja capacitancia, estas líneas son adecuadas para conducir señales de alta frecuencia. Si las ocho entradas para las redes de reloj están ocupadas, estas líneas son adecuadas como alternativa. *Hex lines* son las que conectan una salida de cada tres CLBs (figura 4-25b). Son líneas que ofrecen mayor conectividad que las anteriores, pero un poco menos de capacidad en alta frecuencia. Las *Double lines* conectan todos los otros CLBs (figura 4-25c), lo que las hace conexiones más flexibles. Las *Direct lines* entregan conexiones directas de cada CLB hacia cada uno de sus ocho vecinos (figura 4-25d). Estas líneas son usadas más a menudo para conducir una señal proveniente de un CLB de origen hacia una *Double line*, *Hex line* o *Long line* y desde esa ruta larga hacia otra *Direct line* que llevará la señal hacia el CLB de destino.

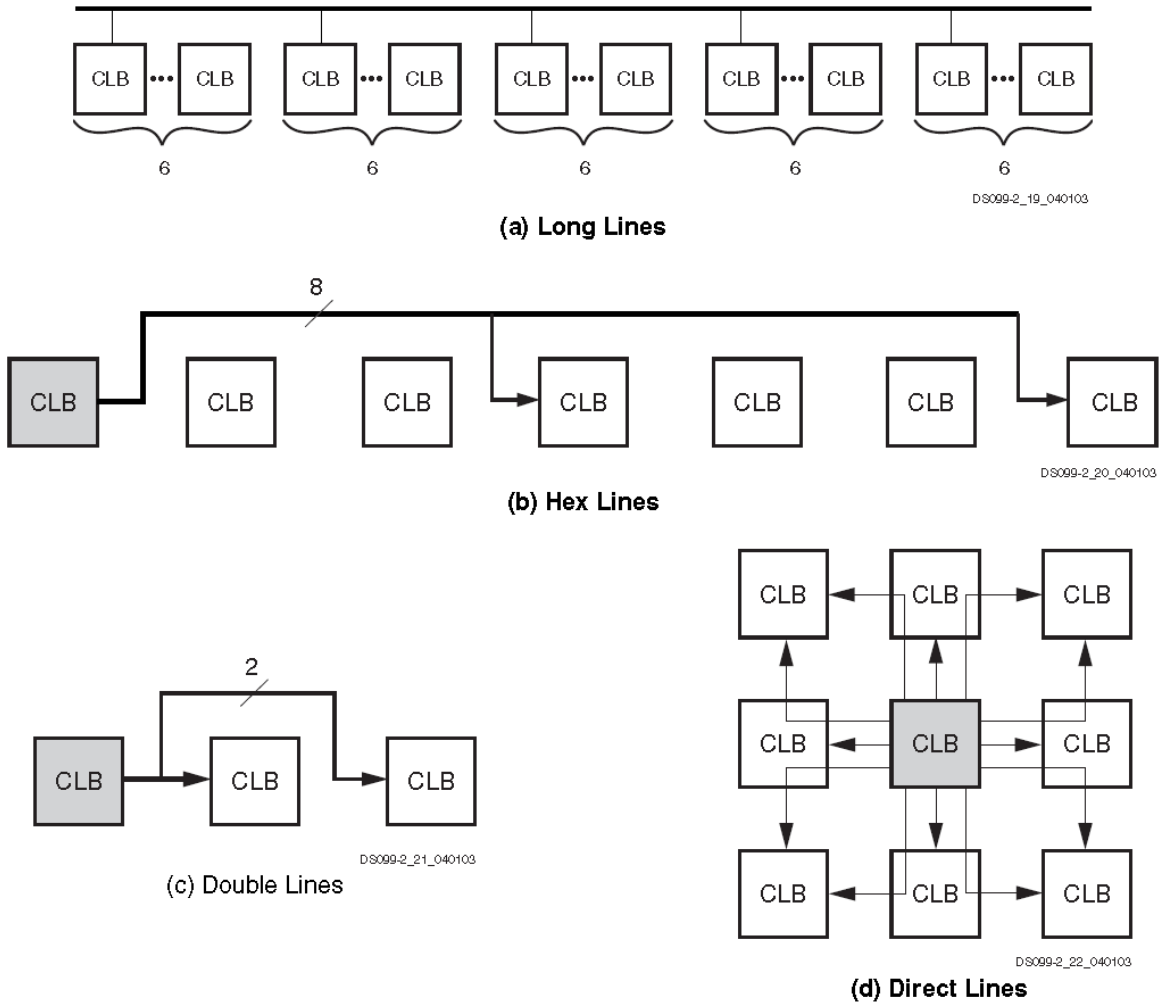


Figura 4-25 Tipos de interconexiones entre CLBs en la Spartan 3.

En la tabla 4-2 comparo los diversos modelos de la familia Spartan 3 y sus principales características.

Tabla 4-2 Principales características de los diferentes modelos de la familia Spartan 3.

Device	XC3S50A/AN	XC3S200A/AN	XC3S400A/AN	XC3S700A/AN	XC3S1400A/AN	XC3S1800A	XC3S3400
System Gates	50000	200000	400000	700000	1400000	18000000	3400000
Logic Cells	1584	4032	8064	13248	25344	37440	53712
CLB	176	448	896	1472	2816	4160	5968
Slices	704	1792	3584	5888	11264	16640	23872
Distribuited RAM Bits	11000	28000	56000	92000	176000	260000	373000
Block RAM Bits	54000	288000	360000	360000	576000	1512000	2268000
In-system Flash Bits	1000000	4000000	40000000	8000000	16000000	-	-
Dedicated Multipliers	3	16	20	20	32	-	-
DSP48A	-	-	-	-	-	84	126
DCM	2	4	4	8	8	8	8
Maximum User I/O	144	248	311	372	502	519	469

Device	System Gates	Equivalent Logic Cells	CLBs	Slice	Distribuited RAM Bit	Block RAM Bit	Dedicated Multiplier	DCM	Maximum User I/O
XC3S100E	100K	2160	240	960	15K	72K	4	2	108
XC3S250E	250K	4508	612	2448	38K	216K	12	4	172
XC3S500	500K	10476	1164	4656	73K	360K	20	4	232
XC3S1200E	1200K	19512	2168	8672	136K	504K	28	8	304
XC3S1600E	1600K	33192	3688	14752	231K	648K	36	8	376

4.4.1.7. Multiplicadores y bloques DSP de las FPGA de Xilinx

Hace ya algunos años Xilinx sacó al mercado la familia Virtex–II que incorpora un multiplicador dedicado de 18 bits con salida de 36 bits llamado MULT18x18S (figura 4-26). Esta combinación aumenta notablemente la velocidad de las multiplicaciones comparada con las familias anteriores basadas en LUTs.

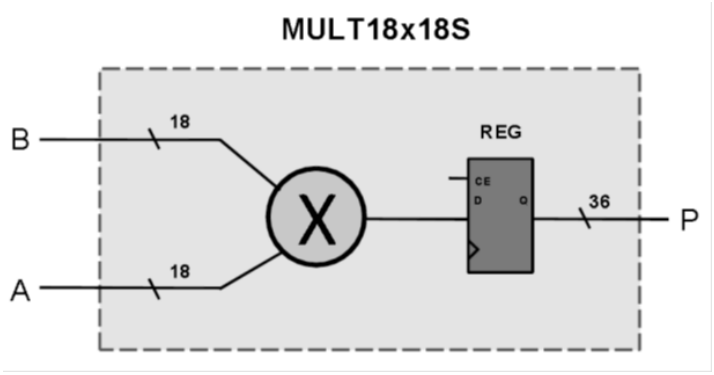


Figura 4-26 Esquema simplificado del Multiplicador de 18x18 bits que incorpora la Virtex–II.

La familias de bajo coste de Spartan, 3A y 3E, se basan en las primeras versiones de la Virtex, e incorporan multiplicadores dedicados de 18 bits con resultados de 36 bits llamados MULT18x18SIO (figura 4-27). Puede alcanzar una frecuencia de trabajo superior a 240 Mhz.

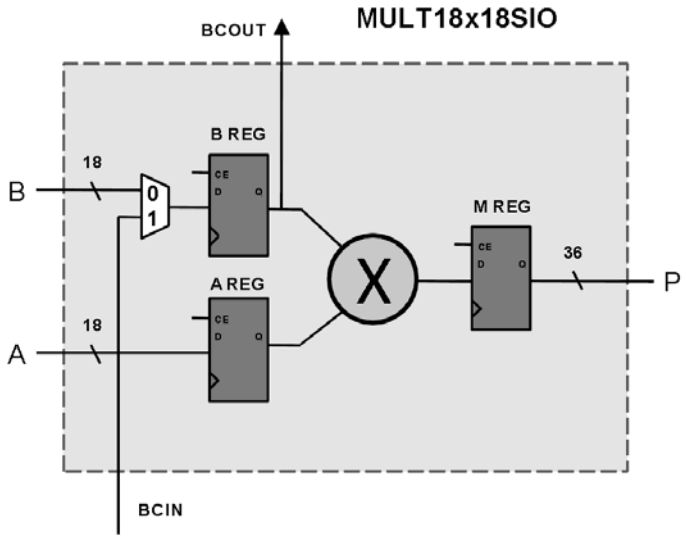


Figura 4-27 Esquema simplificado del Multiplicador de 18x18 bits que incorpora la Spartan–3A/3E.

La Virtex–4 incorpora un nuevo bloque de DSP que es capaz de multiplicar y sumar en el mismo módulo (figura 4-28); se le llamó DSP48 debido a su precisión de salida de 48 bits. Además del sumador, se podían realizar restas y operaciones de rotación para facilitar funciones de escalado de los resultados. Se incorporaron registros intercalados entre las operaciones para implementar una estructura *pipeline* y lograr una frecuencia de trabajo superior a 400 Mhz.

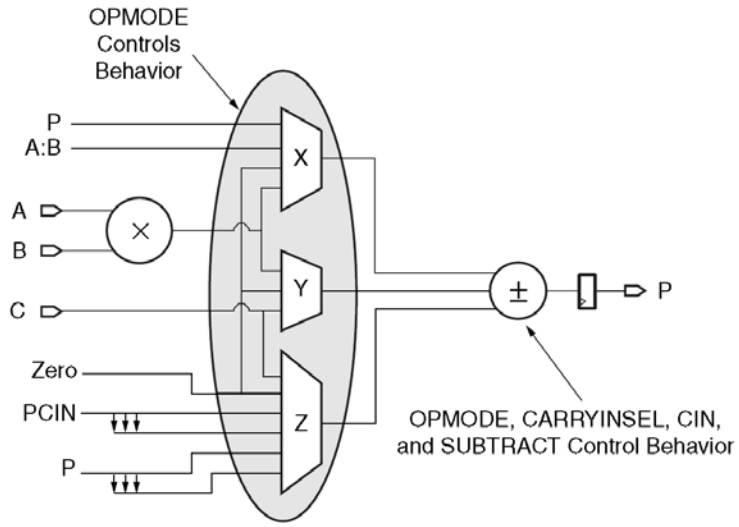


Figura 4-28 Esquema simplificado del móduloDSP48 que incorpora la Virtex–4.

Con la llegada de la Virtex–5 se introdujo el módulo DSP48E ("*Enhanced*"), que está basado en el DSP48 de la Virtex–4. El bloque sumador se modificó para convertirse en un bloque multifunción ALU; entre otras cosas, se soporta detección de saturación, *overflow* y *underflow* (figura 4-29). El comportamiento de esta ALU se puede modificar en tiempo de ejecución mediante códigos a la entrada del mismo. El DSP48E puede alcanzar frecuencias de trabajo superiores a 450 Mhz.

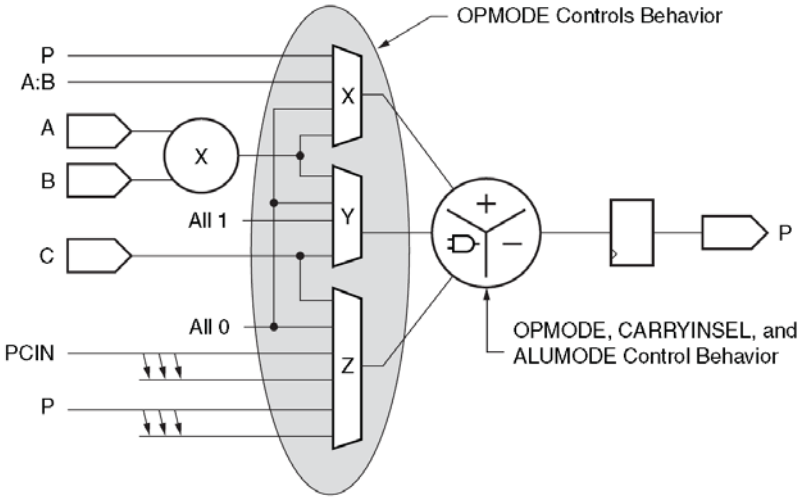


Figura 4-29 Esquema simplificado del módulo DSP48E que incorpora la Virtex–5.

4.4.1.8. Módulo DSP48A de la familia Spartan–3A DSP

La familia Spartan 3A–DSP incorpora el módulo DSP48A descrito en [42], que está basado en el DSP48 de la Virtex–4. El módulo DSP48A soporta completamente las operaciones MAC (*pre–adder*, *multiplier*, *add–accumulate*) (figura 4-30). Estos módulos de DSP dedicados tienen la mejor relación coste/MAC de las FPGAs.

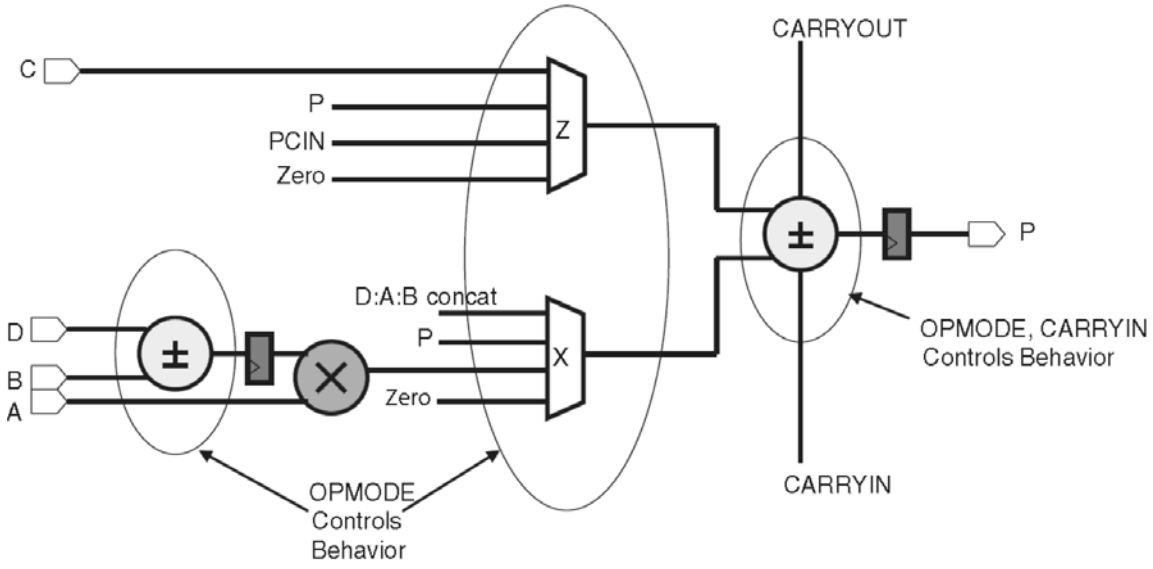


Figura 4-30 Esquema simplificado del DSP48A que incorpora la Spartan 3A–DSP.

En la figura 4-31 se puede ver la evolución de los módulos de DSP de las familias de FPGA de Xilinx (Virtex y Spartan).

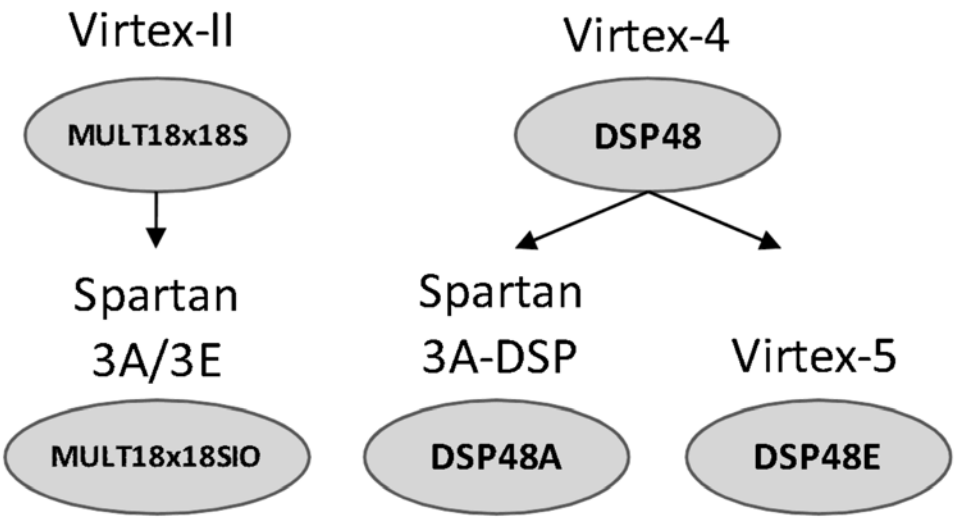


Figura 4-31 Evolución de los módulos DSP de las familias de FPGAs de Xilinx.

En la tabla 4-3 puede verse una comparativa en modo resumido entre los diferentes módulos DSP48 de las familias de Xilinx.

Tabla 4-3 Comparativa entre los distintos módulos DSP48 de las FPGAs de Xilinx

Función	DSP48	DSP48E	DSP48A	Beneficios
Multiplicador	18x18	25x18	18x18	Se reducen los recursos necesarios para algoritmos de DSP
Pre–sumador	no	no	si	Reducción del camino crítico para filtros FIR
Entradas en cascada	1	2	1	Permite conectar en cascada los módulos para filtros grandes
Salidas en cascada	si	si	si	Permite conectar en cascada los módulos para filtros grandes
Entrada dedicada C	no	si	si	Permite realizar operaciones matemáticas con 3 entradas
Entradas de sumador	3x48 bits	3x48 bits	2x48 bits	Soporte para suma y acumulación
Modo dinámico	si	si	si	Soporte para múltiples operaciones configurables en tiempo de ejecución
Funciones lógicas ALU	no	si	no	Similar a la ALU de un microprocesador
Detección de patrones	no	si	no	Detección de saturación, overflow y otros
Soporte de múltiples ALU	no	si	no	Permite realizar en paralelo múltiples operaciones con la ALU
Señales de carry	Carry in	Carry in–out	Carry in–out	Carry rápido, útil para interconectar DSPs

En la tabla 4-4 se muestran los dos dispositivos de la familia Spartan 3A–DSP que incorporan el módulo DSP48A. Como se ha comentado antes, es una familia de bajo coste que incorpora altas prestaciones en su módulo de DSP.

Tabla 4-4 Dispositivos de la familia Spartan 3A DSP y sus características.

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM Bits	Block RAM Bits	DSP48As	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3SD1800A	1800K	37,440	88	48	4,160	16,640	260K	1,512K	84	8	519	227
XC3SD3400A	3400K	53,712	104	58	5,968	23,872	373K	2,268K	126	8	469	213

El dispositivo XC3SD1800A, está distribuido en 4 columnas de elementos DSP48A, sumando un total de 84. El XC3SD3400A tiene 5 columnas de elementos DSP48A y suma un total de 126.

En la figura 4-32 puede verse en detalle la ubicación de los distintos módulos (DSP48A, BRAM, DCM y CLBs) dentro de una FPGA de la Familia Spartan 3A–DSP. Cómo es lógico, el DSP48A está ubicado en un lugar estratégico rodeado de CLBs y memoria RAM para realizar operaciones de DSP y acceder rápidamente a los puertos de entrada salida (IOBs). En comparación con la anterior familia Virtex–II, se dispone de recursos de interconexión independientes lo cual permite doblar el ancho de banda disponible entre los distintos elementos.

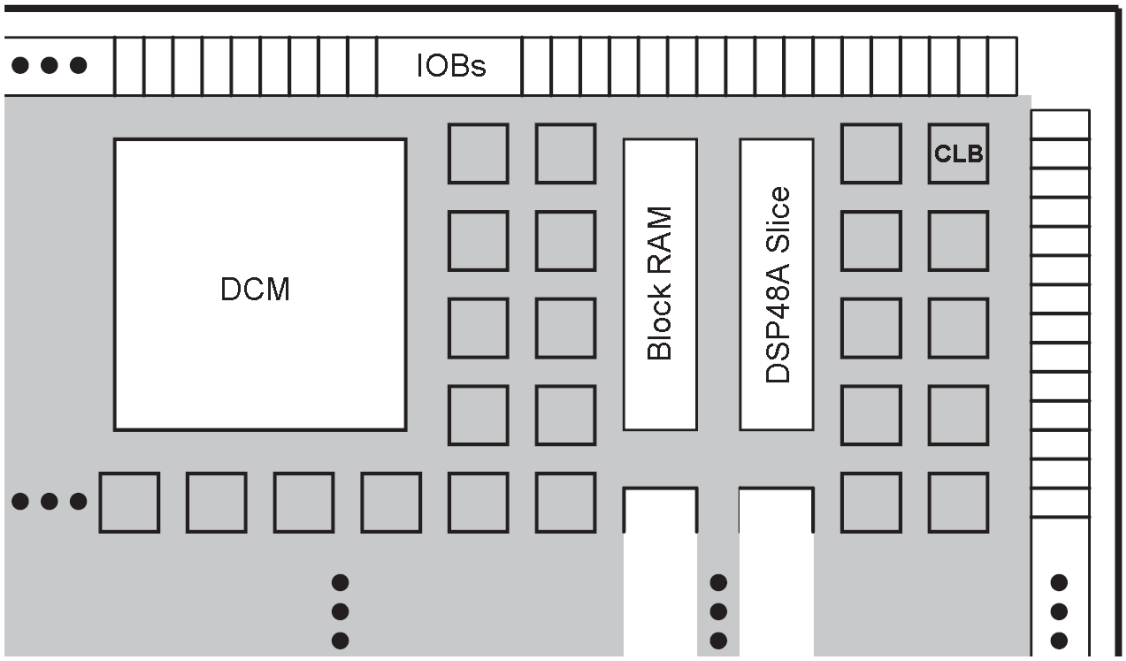


Figura 4-32 Detalle de la ubicación de los módulos DSP48A en una FPGA de Xilinx.

Cada DSP48A tiene un pre–sumador, que acepta a su entrada 18 bits en complemento a dos, y genera a la salida resultados de 18 bits también en complemento a dos. A continuación del pre–sumador, se encuentra el multiplicador con dos entradas de complemento a dos de 18 bits, que genera a su salida resultados de 36 bits en complemento a dos, y que extendiendo el signo se convierte en 48 bits. Seguido del multiplicador, se encuentra otro sumador que opera en 48 bits complemento a dos.

A continuación se destacan las características más importantes del módulo DSP48A:

- Multiplicador dedicado de 18x18 bits con salida de 36 bits extendiendo el signo a 48.
- Arquitectura *pipeline* para funcionar por encima de los 250 Mhz.
- Acumulador de 48 bits con salida registrada para realimentación del mismo.
- Integración de multiplicador y sumador para rapidez en operaciones.
- Pre–sumador de 18 bits para implementación eficiente de filtros simétricos.
- Posibilidad de conectar módulos en cascada para múltiples operaciones.

En la figura 4-33 se muestran los componentes internos del módulo DSP48A.

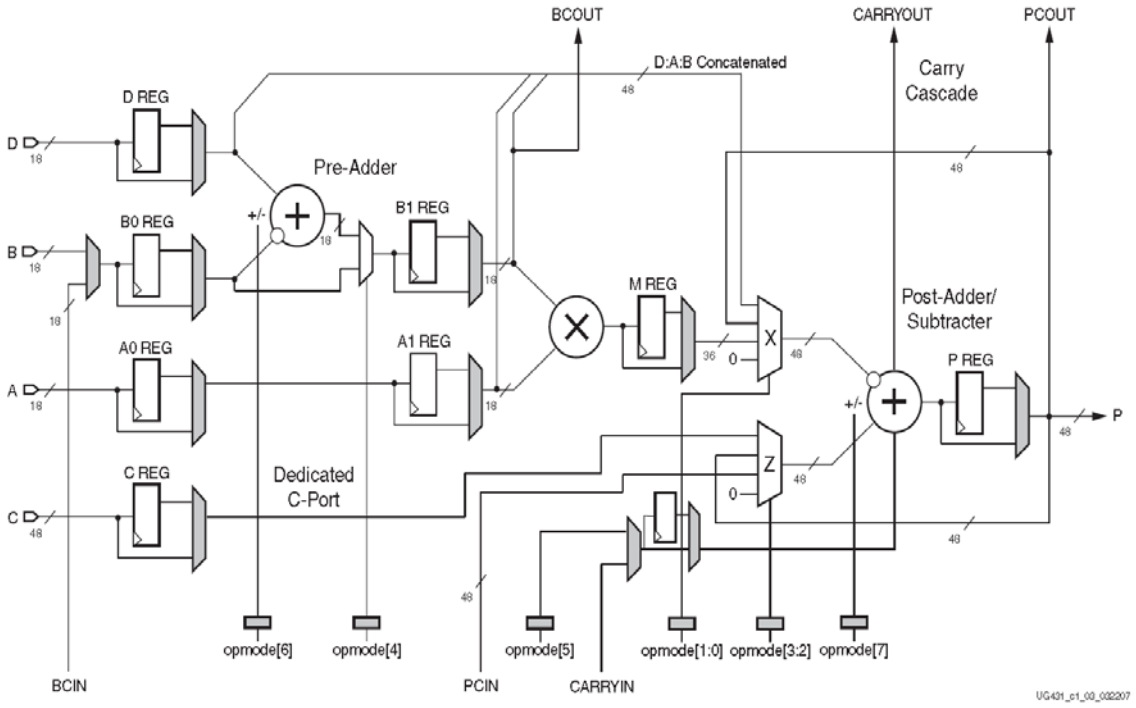


Figura 4-33 *Slice* del DSP48A.

En la tabla 4-5 se explican cada uno de los puertos del módulo DSP48A:

Tabla 4-5 Lista de puertos disponibles del DSP48A.

Nombre	Dirección	Tamaño	Función
			Puertos de datos
A	IN	18	Entrada al multiplicador, pos-sumador/restador, se configura con OPMODE[3:0]
B	IN	18	Entrada al multiplicador, pre-sumador/restador, pos-sumador/restador, se configura con OPMODE[1:0] y OPMODE[4]
C	IN	48	Entrada al pos-sumador/restador
D	IN	18	Entrada al pre-sumador/restador
Carry IN	IN	1	Entrada de carry externo hacia el pos-sumador/restador
P	OUT	48	Salida de datos primaria
Carry OUT	OUT	1	Salida de carry del pos-sumador/restador
			Puertos de control
CLK	IN	1	Entrada de reloj
OPMODE	IN	8	Entrada de control para seleccionar la operación aritmética
			Puertos en cascada
PCIN	IN	48	Entrada en cascada conectada al pos-sumador/restador
BCIN	IN	18	Entrada alternativa a B, conectada al pre-sumador/restador, multiplicador, pos-sumador/restador
PCOUT	OUT	48	Salida en cascada del puerto P, al PCIN de otro DSP48A
BCOUT	OUT	18	Salida en cascada del puerto B, al B/BCIN de otro DSP48A
			Puertos de Reset y Habilitación

RST{x}	IN	1	Reset de registros A, B, C, D, M, P, carry-in y OPmode
CE{x}	IN	1	Habilitación de reg. A, B, C, D, M, P, carry-in y OPmode

En la tabla 4-6 se detalla el registro de configuración OPMODE[7:0].

Tabla 4-6 Descripción del registro de configuración OPMODE.

Nombre	Función
OPMODE[1:0]	Multiplexor X (entrada al pos-sumador/restador)
	0: ceros (deshabilitación del pos-sumador/restador)
	1: salida del multiplicador
	2: señal P (acumulador)
	3: señales D, B, A concatenadas
OPMODE[3:2]	Multiplexor Z (entrada al pos-sumador/restador)
	0: ceros (deshabilitación del pos-sumador/restador, propagación del multiplicador)
	1: señal PCIN
	2: señal P (acumulador)
	3: entrada C
OPMODE[4]	Uso del pre-sumador/restador
	0: propagar la entrada directamente al multiplicador
	1: sumar o restar B y D; aplicar el resultado al multiplicador.
OPMODE[5]	Forzado del carry-in
OPMODE[6]	Configuración del pre-sumador/restador
	0: actúa como sumador
	1: actúa como restador
OPMODE[7]	Configuración del pos-sumador/restador
	0: actúa como sumador
	1: actúa como restador

A través de los atributos del DSP48A es posible configurar el registro de las entradas, resultados de operaciones internas o salidas. Como puede verse en la figura 4-20, los registros disponibles son: A0REG, A1REG, B0REG, B1REG, CREG, DREG, MREG, PREG y CARRYINREG.

Para describir el modo de operación del módulo DSP48A según la figura 4-30, este módulo puede dividirse en tres bloques básicos: pre-sumador/restador, multiplicador y post-sumador/restador. Podemos plantear la ecuación siguiente que describe el funcionamiento genérico:

$$P = C \pm (A \times (D \pm B) + \text{CARRY IN})$$

A partir de ésta ecuación, pueden derivarse otras con las que se obtienen los modos más importantes de funcionamiento descritos en la tabla 4-7.

Tabla 4-7 Principales modos de funcionamiento del DSP48A.

Modo	Ecuación
Multiplicador	± (A x B + CARRY IN)
Pre-sumador/Multiplicador	± (A x (D ± B) + CARRY IN)
Pre-sumador/Sumador en cascada	PCIN ± D : A : (D ± B) + CARRY IN
Pre-sumador/Multiplicador/Sumador en cascada	PCIN ± (A x (D ± B) + CARRY IN)
Pre-sumador/Multiplicador/Sumador realimentado	P ± (A x (D ± B) + CARRY IN)
Sumador de 48 bits con C	C ± D : A : B + CARRY IN
Pre-sumador/Multiplicador/Sumador con C	C ± (A x (D ± B) + CARRY IN)
Pre-sumador/Sumador de 48 bits	C ± D : A : (D ± B) + CARRY IN

El DSP48A es un módulo muy potente para la realización de diversas operaciones matemáticas, las más importantes, la suma y la multiplicación de números. Para el objeto de investigación de esta tesis, se utilizará principalmente la operación MAC. La disponibilidad de éste módulo permite desarrollar prototipos de muchas aplicaciones donde la frecuencia de trabajo y el ancho de palabra son factores críticos.

4.4.2. Arquitectura de la FPGA Spartan 6 de Xilinx

Esta familia de FPGA sigue la línea de la Spartan–3 que proporciona un buen rendimiento con alto volumen de producción a muy bajo coste. Los modelos disponibles de esta familia proporcionan una gama de celdas lógicas que van desde 3840 hasta 147.443 y un consumo de energía reducido en un 50% en comparación con el de la familia Spartan-3. Estas FPGAs son fabricadas en una tecnología de 45 nm. Dispone de dos subfamilias: Spartan–6 LX: de lógica optimizada y Spartan–6 LXT con conectividad serie de alta velocidad.

Los principales cambios con respecto a la generación anterior son los siguientes:

- LUT de 6 entradas.

- 16 redes globales de reloj

- Los bloques de control de memoria (MCB) interacciona con un solo chip DRAM (o DDR, DDR2 o DDR3 LPDDR) con velocidad de acceso de hasta 800 Mb/s. El MCB dispone de pistas dedicadas para predefinir las E/S de la FPGA. El controlador de memoria proporciona una interfaz multi–puerto con la lógica de la FPGA y puede ser conectado con una memoria DRAM externa a 4, 8 o 16 bits. En muchas aplicaciones un MCB proporciona a una interfaz de DRAM más rápido que el bus de datos convencional.

- La CMT (bloques de gestión del reloj) que contiene dos bloques DCM y PLL.

- Los transceptores de baja potencia son un transmisor y un receptor combinado para funcionar a una velocidad de datos máxima de 3,2 Gb/s. Estos dos circuitos son independientes y utilizan PLLs separadas para multiplicar las frecuencias de entrada. El transmisor es un convertidor de paralelo a serie con una capacidad de conversión de 8, 10, 16 o 20 bits. El receptor es un convertidor de serie a paralelo, que cambia la señal diferencial de serie en un flujo de bits de las palabras en paralelo, cada uno de 8, 10, 16 o 20 bits.

- PCI Express es una interfaz de punto a punto serie estándar y su especificación define una velocidad de transferencia de 2,5 Gb/s por canal, por dirección (transmisión y recepción). Los dispositivos Spartan–6 incluyen un bloque de punto final integrado para la tecnología PCI Express que cumple con sus especificaciones. Este bloque es configurable y actúa como uno solo en los extremos del canal. Interfaces de este bloque con los transceptores GTP para la serialización/deserialización con bloques y RAM de memoria intermedia de datos.

Voy a hacer una breve introducción a la arquitectura de la Spartan 6 de Xilinx [43].

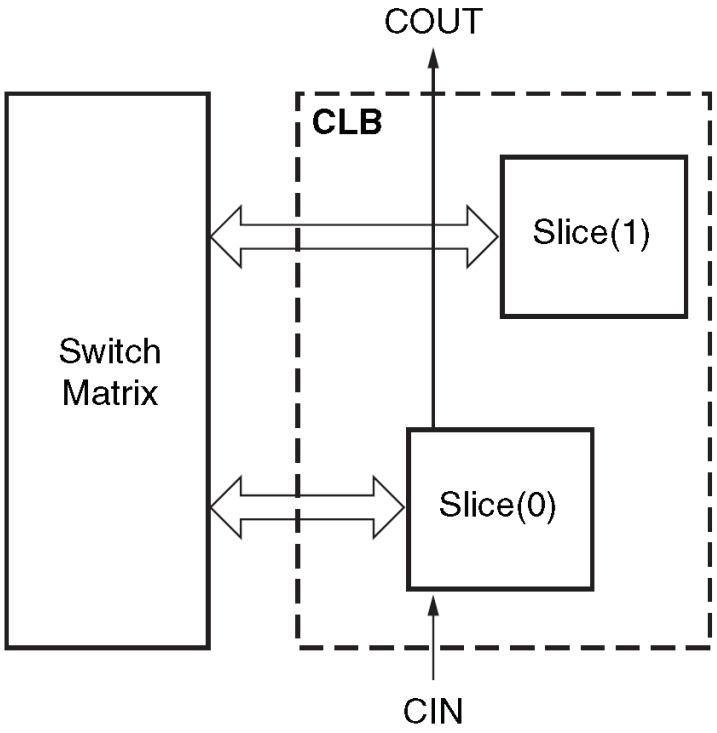


Figura 4-34 Diagrama general de un CLB de la Spartan 6 de Xilinx.

CLB: corresponden a los bloques lógicos configurables básicos y están compuestos de dos *slices* no conectados entre sí. Cada *slice* contiene cuatro LUT 6 además de otros recursos. En el caso de una Spartan 6 existen tres tipos de slice que contienen diferentes recursos llamados SLICEX, SLICEL y SLICEM. Los tres tipos de slice contienen como elementos comunes 4 LUT de 6 entradas y flip–flop para almacenamiento. La tabla 4-8 resume los recursos lógicos que contiene cada tipo de slices. Cada CLB se compone de un SLICEX y un SLICEL o SLICEM según corresponda y estos CLBs se conectan entre sí mediante una matriz *switcheada* de interconexión; no obstante, cada CLB posee algunas interconexiones directas consigo mismo y con algunos CLBs contiguos.

Tabla 4-8 Tipos de slices en la Spartan-6 y sus características

Característica	SLICEX	SLICEL	SLICEM
LUT de 6 entradas	√	√	√
8 flip-flops	√	√	√
Multiplexores de función amplia		√	√
Lógica de acarreo		√	√
RAM distribuida			√
Registros de desplazamiento			√

En las figuras 4-35 y 4-36 muestro la matriz de CLBs y su interconexión.

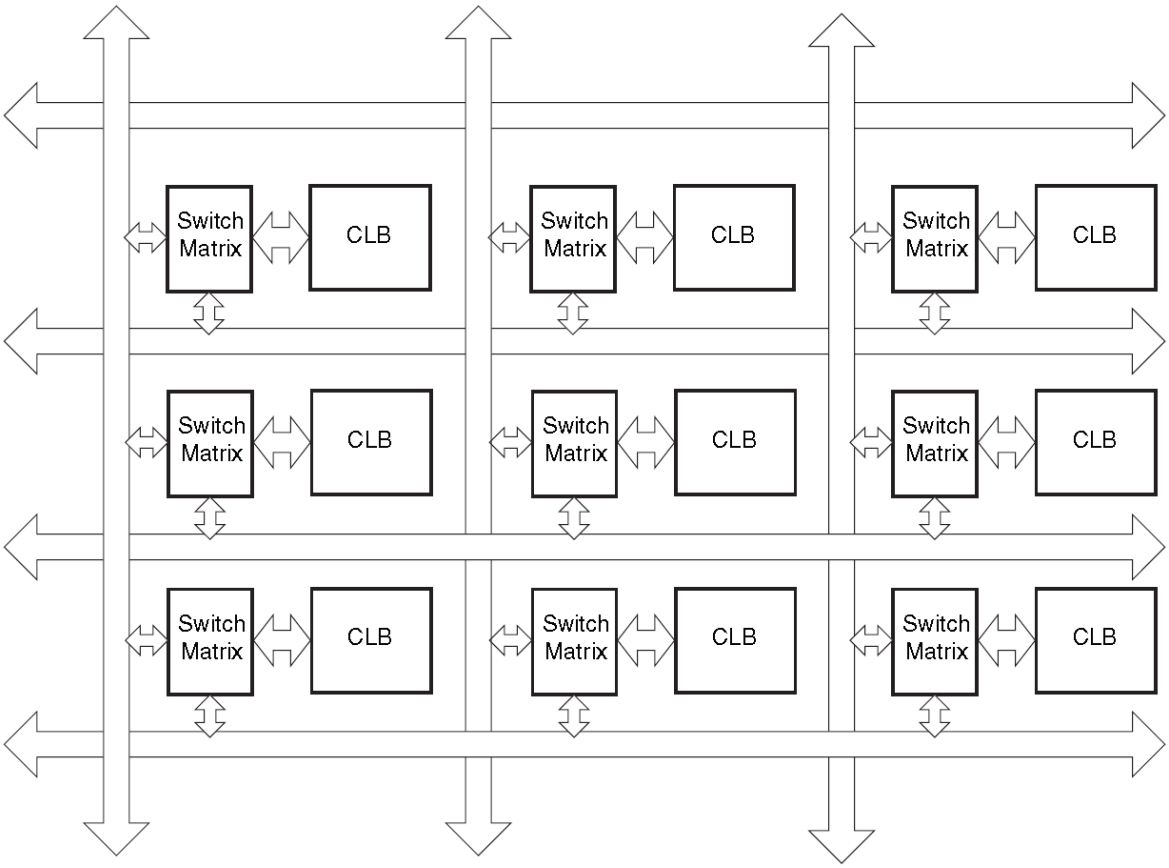


Figura 4-35 Matriz de CLBs y canales de interconexión.

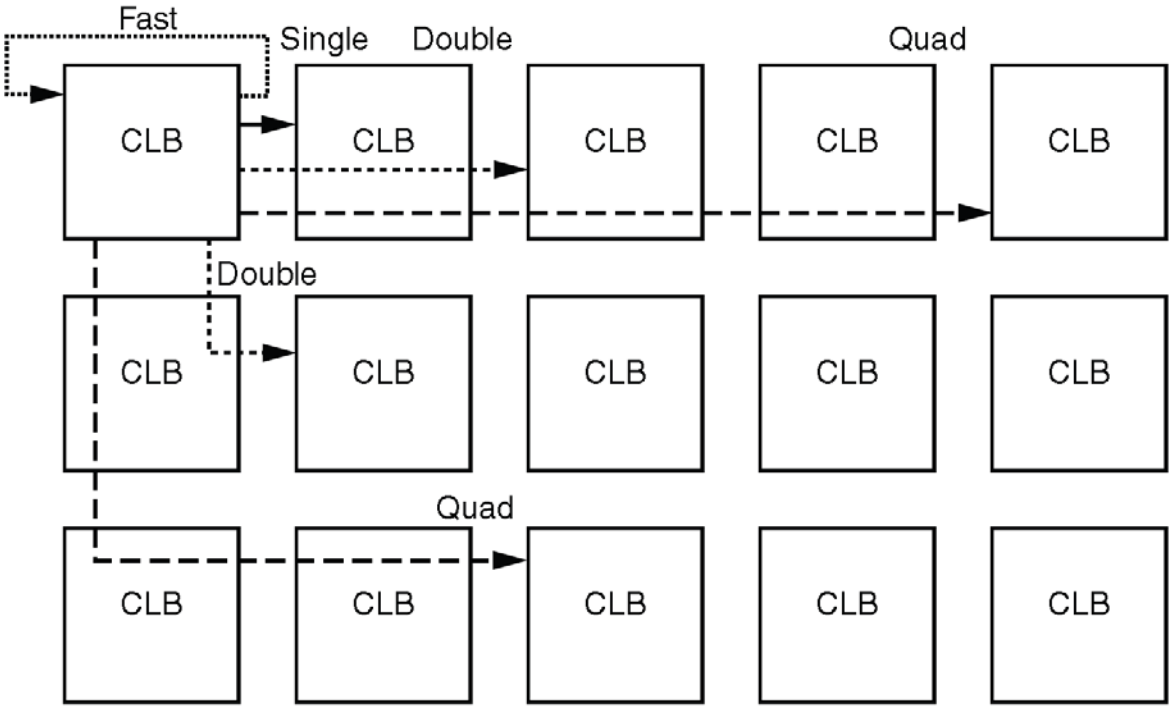


Figura 4-36 Ejemplos de tipos de interconexión de CLBs.

DSP slices: Son bloques de alto rendimiento y bajo consumo de potencia, específicos para la realización de cálculos matemáticos. Se componen de un multiplicador dedicado de 18 bits en complemento a 2 y un acumulador de 48 bits, siendo capaz de operar hasta los

390 MHz. Con este bloque DSP48A1 se pueden realizar multiplicaciones, preadiciones, acumulación, contadores sincrónicos y *barrel shifting*. Lo comentaré con más detalle en el apartado 4.4.2.1

CMT: Son bloques de manejo de reloj para distribución, síntesis, desfase y otros. Cada CMT está compuesto por dos DCMs, una PLL y varias líneas de distribución, tanto locales como globales o de I/O.

BRAM: Son bloques internos de memoria RAM de alta velocidad. Los bloques son de 18 Kbits de doble puerto "real", que tan solo comparten los datos almacenados y, por tanto, pueden operar de manera simultánea.

MCB: Es un bloque controlador de memoria específico para conectar la FPGA con un chip de DRAM, pudiendo alcanzar velocidades de hasta 800 Mbits, mediante el uso de las rutas dedicadas.

IOB: Corresponde al bloque configurable de entrada/salida presente en cada pin. Soporta 18 estándares de I/O de un terminal y 8 estándares diferenciales.

Las FPGAs Spartan 6 y Virtex 6 contienen LUT que pueden ser configuradas como LUT con seis entradas y dos salidas o bien, como dos LUT con 5 entradas y una salida por LUT. La figura 4-37 muestra una estructura simplificada de las LUT 6.

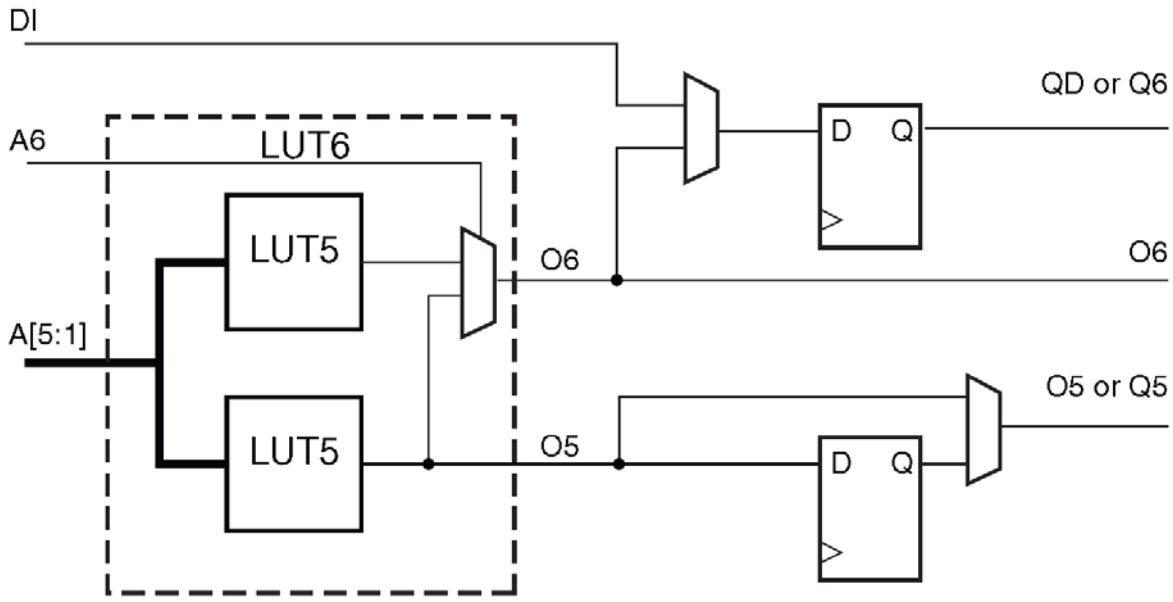


Figura 4-37 Vista simplificada de la conectividad de una LUT6 simple de la Spartan 6 de Xilinx.

En la tabla 4-9 comparo los diversos modelos de la familia Spartan 6 y sus principales características.

Tabla 4-9 Principales características de los diferentes modelos de la familia Spartan 6

Device	Logic Cells	Slices	Flip-Flops	Max Distribuited RAM	DSP481A Slices	18Kb RAM Blocks	Max RAM Blocks
XC6SLX4	3840	600	4800	75	8	12	216
XC6SLX9	9152	1430	11440	90	16	32	576
XC6SLX16	14579	2278	18224	136	32	32	576
XC6SLX25	24051	3758	30064	229	38	52	936
XC6SLX45	43661	6822	54576	401	58	116	2088
XC6SLX75	74637	11662	93296	692	132	172	3096
XC6SLX100	101261	15822	126576	976	180	368	4824
XC6SLX150	147443	23038	184304	1355	180	268	4824
XC6SLX25T	24051	3758	30064	229	38	52	936
XC6SLX45T	43661	6822	54576	401	58	116	2088
XC6SLX75T	74637	11662	93296	692	132	172	3096
XC6SLX100T	101261	15822	126822	976	180	268	4824
XC6SLX150T	147443	23038	184304	1355	180	268	4824

Device	CMTs	Memory Controller Block	Endpoint Blocks for PCI Express	Maximum GTP Transceivers	Total I/O Bank	Max User I/O
XC6SLX4	2	0	0	0	4	132
XC6SLX9	2	2	0	0	4	200
XC6SLX16	2	2	0	0	4	232
XC6SLX25	2	2	0	0	4	266
XC6SLX45	4	2	0	0	4	358
XC6SLX75	6	4	0	0	6	408
XC6SLX100	6	4	0	0	6	480
XC6SLX150	6	4	0	0	6	576
XC6SLX25T	2	2	1	2	4	250
XC6SLX45T	4	2	1	4	4	296
XC6SLX75T	6	4	1	8	6	348
XC6SLX100T	6	4	1	8	6	498
XC6SLX150T	6	4	1	8	6	540

4.4.2.1. Módulo DSP48A1 de la familia Spartan–6 DSP

Actualmente se sigue trabajando en la optimización de los módulos DSP para aumentar el rango de sus parámetros, ofrecer mayor escala de integración, menor consumo, menor coste y nuevas funcionalidades. Como sucesor del módulo DSP48A de la Spartan 3, se encuentra el DSP48A1 [44] perteneciente a la familia Spartan-6 de Xilinx y que salió al mercado en agosto de 2009. Principalmente se destaca por estar más optimizado en el cálculo de las operaciones y consumir menos potencia.

El *slice* DSP48A1 se deriva del DSP48A que contiene la FPGA Extended Spartan–3A. Muchos algoritmos DSP utilizan de forma mínima los recursos de una FPGA de propósito

general, obteniendo bajo consumo, alto rendimiento, y una utilización eficiente del dispositivo.

Los *slice* DSP48A1 pueden implementar muchas funciones independientes, tales como multiplicador, multiplicador-acumulador [8], pre-sumador/restador seguido de un MAC, multiplicador seguido por un sumador, multiplexores, comparador de magnitud, o contadores.

En principio, el DSP48A1 dispone de un pre–sumador de entrada de 18bits seguido de un multiplicador en complemento a 2 de 18x18 bit y un sumador/restador/acumulador de 48 bits para la extensión de signo, que es una función muy utilizada en el procesamiento digital de señales. Por otro lado tiene algunas características que mejoran la utilidad, versatilidad y velocidad de este bloque aritmético. El hecho de disponer de una estructura *pipeline* para los operandos de entrada, los productos intermedios y las salidas del acumulador mejora el rendimiento. El bus interno de 48 bits permite conectar los *slices* de manera prácticamente ilimitada.

Una de las características más importantes es la posibilidad de unir en cascada el resultado de un *slice* DSP48A1 al siguiente sin utilizar el rutado general. Este hecho proporciona alto rendimiento con bajo consumo de energía para la realización de filtros DSP de cualquier longitud. Unidos en cascada pueden realizar operaciones matemáticas con operadores mucho más extensos, convolución y filtros, entre otras operaciones aritméticas complejas sin utilizar CLB´s comunes de la FPGA. Otra ventaja para la realización de filtros es la posibilidad de unir en serie un dato de *slice* a *slice*.

El puerto de entrada C permite realizar funciones matemáticas de tres entradas, como por ejemplo una suma de tres entradas conectando en serie el pre–sumador con el post–sumador, o también una multiplicación seguida de una suma (operación MAC). Además la entrada D puede ser utilizada con el pre–sumador para reducir el número de *slices* DSP48A1 utilizados en filtros simétricos (figura 4-38).

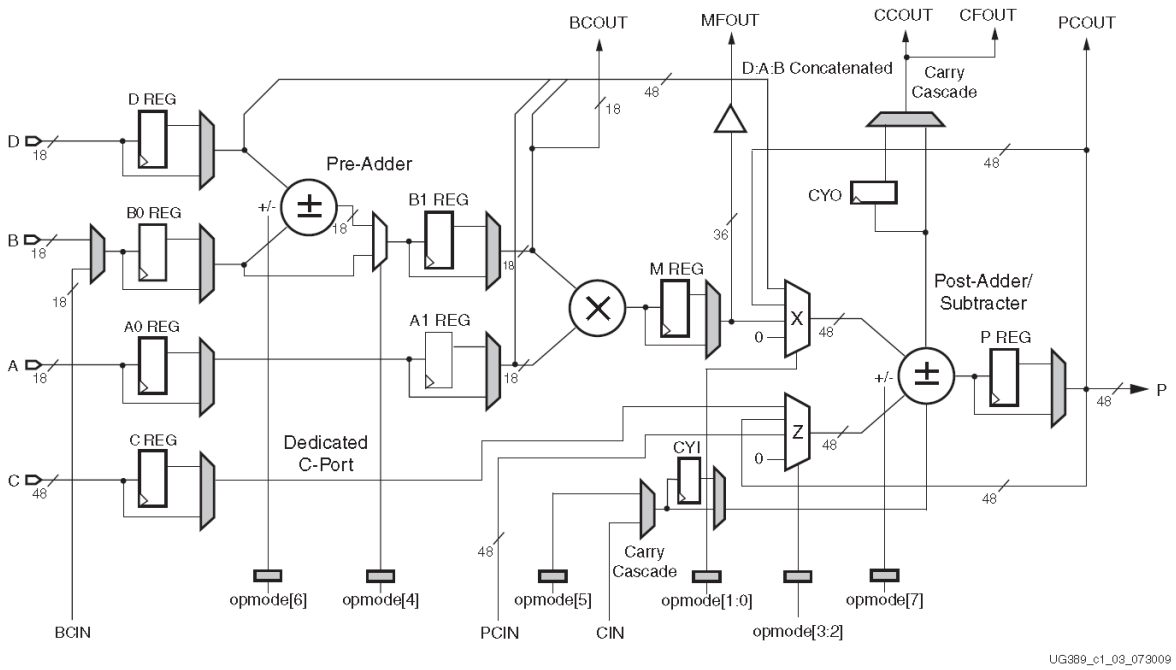


Figura 4-38 *Slice* del DSP48A1.

Los *slices* DSP48A1 están organizados como columnas verticales DSP, dentro de cada columna DSP, cada *slice* está combinado con lógica extra y enrutamiento. Esta configuración en columnas verticales es ideal para conectar un *slice* DSP con sus *slices* adyacentes, por tanto uniendo en serie estos bloques facilita la implementación del algoritmos sistólicos DSP.

La famila Spartan 6 ofrece una alta relación de *slices* DSP48A1 con respecto a la lógica disponible, lo que la hace aconsejable para aplicaciones que necesiten mucho cálculo matemático. En la tabla 4-10 muestro los *slices* de este tipo para cada dispositivo de la mencionada familia.

Tabla 4-10 Número de slices DSP48A1 de los diferentes modelos de la familia Spartan 6

Dispositivo	Total *slices* DSP48A1 por dispositivo	Número de columnas DSP48A1 por dispositivo	Número de *slices* DSP481A por columna
XC6SLX4	8	1	8
XC6SLX9	16	1	16
XC6SLX16	32	2	16
XC6SLX25	38	2	18/20
XC6SLX45	58	2	30/28
XC6SLX75	132	3	44/40/48
XC6SLX100	180	4	48/44/40/48
XC6SLX150	180	4	48/44/40/48
XC6SLX25T	38	2	18/20
XC6SLX45T	58	2	30/28
XC6SLX75T	132	3	44/40/48
XC6SLX100T	180	4	48/44/40/48
XC6SLX150T	180	4	48/44/40/48

Cada *slice* DSP48A1 dispone de un presumador de 18 bit seleccionable que admite dos entradas de 18 bits en complemento a dos y genera el resultado de 18 bits en complemento a dos. El presumador es seguido de un multiplicador de dos entradas, multiplexores y un postsumador/restador de dos entradas. El multiplicador acepta dos entradas de 18 bits en complemento a dos y da un resultado de 36 bits en complemento a dos. La salida de 36 bits del multiplicador (registrada en M_register, MFOUT) puede ser rutada directamente a la lógica de la FPGA. Al resultado se le extiende el signo a 48 bits y puede opcionalmente enviarse al postsumador/restador. Este acepta dos operandos de complemento a dos de 48 bits y produce un resultado de 48 bits en complemento a dos. Las funciones de alto nivel DSP pueden realizarse conectando los *slices* individuales DSP48A1 en una columna DSP.

Los aspectos más destacados del *slice* DSP48A1 son los siguientes:

- Un presumador de dos entradas para la implementación eficiente de filtros.
- Un multiplicador en complemento a dos con dos entradas de 18 bits y resultado de 36 bits con extensión de signo a 48 bits.
- Un postsumador/restador con dos entradas de 48 bits con opción de acumular y registrar el dato obtenido.
- Modos de operación dinámicos controlados por el usuario para adaptar las funciones del *slice* DSP48A1 de ciclo en ciclo de reloj.
- Bus B de 18 bits que conectado en serie apoya a la propagación de las muestras de entrada.
- Bus P de 48 bits que conectado en serie apoya a la propagación de a la salida de los resultados parciales.
- Gestión eficiente del acarreo (conexión en cascada, posibilidad de registro y rutado a la lógica del usuario).
- La salida del multiplicador de 36 bit puede conectarse directamente a la lógica del usuario.

- Las opciones del *pipeline* para mejorar el rendimiento de las señales de control y de datos se pueden seleccionar mediante bits de configuración.
- El puerto de entrada C es utilizado normalmente para la operación MAC, suma de dos operandos de gran ancho de palabra o técnicas de redondeo flexibles.
- Señales de habilitación del reloj y *reset* separadas para registros de control y datos.
- Registros de entrada/salida, para garantizar el máximo rendimiento del reloj sin coste en área.

A continuación se describen brevemente las ecuaciones lógicas más usuales que puede realizar el módulo DSP48A1. En su forma más básica, la salida del postsumador/restador es función de sus tres entradas: el multiplexor superior (Z), la lógica de acarreo (CIN) y la salida del multiplicador (X) La señal CIN y la salida del multiplexor X siempre se suman, y este resultado se sumará (o restará) a la salida del multiplexor superior (Z).

$$\text{Adder Out} = (\text{Z} \pm (\text{X} + \text{CIN}))$$

En la ecuación siguiente se muestra un uso típico donde se multiplican A y B, y este resultado es sumado (o restado) al registro C. Seleccionando la función multiplicador del multiplexor X, su salida es dirigida al postsumador/restador. Al resultado del multiplicador de 36 bits se le añade una extensión de signo a 48 bits antes de ser enviado al postsumador/restador.

$$\text{Adder Out} = \text{C} \pm (\text{A} \times \text{B} + \text{CIN})$$

En la expresión siguiente, se describe otra utilización del *slice* donde B y D son sumados (o restados) en el presumador/restador, el resultado es multiplicado por A y el resultado de este producto es sumado con C en el postsumador/restador. Esta ecuación facilita la operación MAC utilizada en la convolución.

$$\text{Adder Out} = \text{C} \pm (\text{A} \times (\text{D} \pm \text{B}) + \text{CIN})$$

La figura 4-39 muestra el *slice* DSP48A1 de forma simplificada.

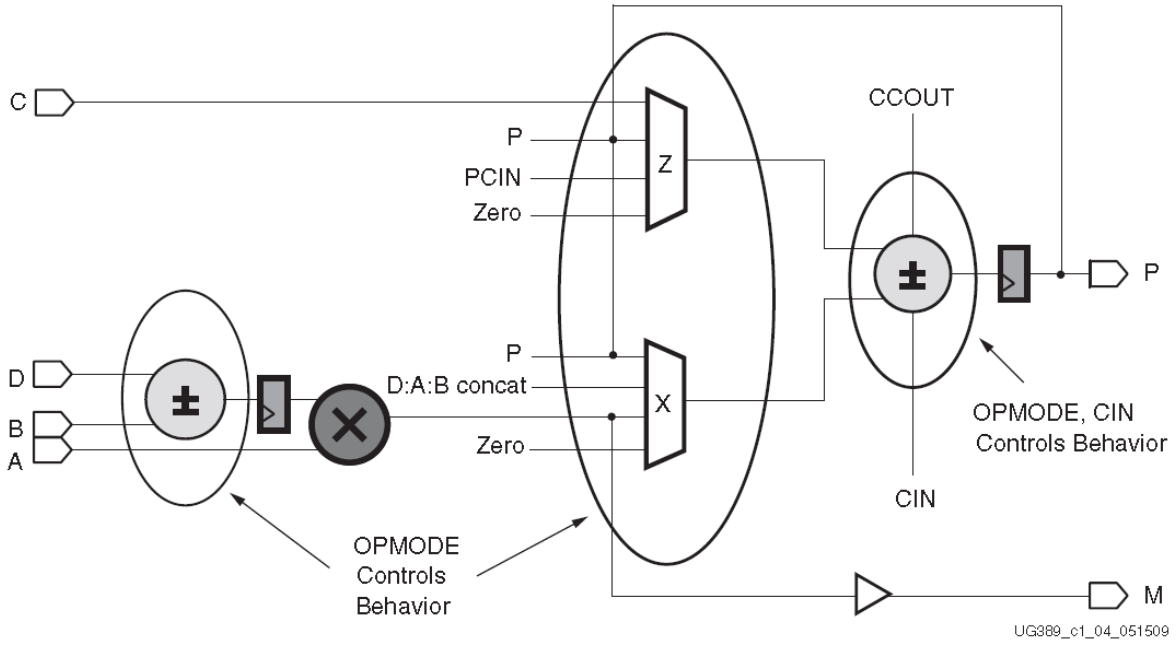


Figura 4-39 Modelo simplificado del *Slice* del DSP48A1.

Los ocho bits del registro OPMODE controlan la selección de suma/resta del presumador/restador y del postsumador/restador, además también controlan los caminos de datos de 48 bits que están conectados a las dos entradas del postsumador/restador y la entrada B al multiplicador. En todos los casos los datos de entrada de 36 bits a los multiplexores son extendidos en signo, formando un dato de entrada de 48 bits al postsumador/restador. Al basarse en operandos de 36 bits y acumulador de 48 bits, los bits de guarda son 12 (por ejemplo pueden ser utilizados para evitar el desbordamiento o extensión de signo). Por lo tanto, el número de posibles operaciones de multiplica–acumula antes de que ocurra desbordamiento es 4096.

4.5. Ventajas de las FPGAs respecto a los DSPs

Resumiendo, una FPGA es un dispositivo de lógica programable en el que existen dentro un altísimo número de elementos básicos de lógica electrónica (como puertas, biestables, etc.) y por encima de ellas hay unas matrices de interconexión que pueden programarse. Se usan lenguajes como VHDL, Verilog, ABEL, y otros para poder describir la aplicación que se quiera implementar. La descripción en estos lenguajes puede estar muy cercana al *hardware* o puede implementarse a más alto nivel (una descripción más algorítmica que de implementación), puesto que los compiladores de estos lenguajes acaban sabiendo realizar los ajustes necesarios en la matriz de conexiones.

Generalmente si se implementa un diseño con FPGA es porque se necesitan una prestaciones críticas, que no es fácil asegurar mediante la programación de un procesador (microprocesador, microcontrolador, DSP, o del tipo que sea). Es por ello que al final en el uso de FPGAs se suelen hacer especificaciones todavía cercanas al mundo del diseño *hardware*, aunque con la enorme ventaja de que se realiza por lenguaje y volcado al dispositivo, con lo que es posible el rediseño y la corrección de errores por reprogramación.

Un DSP es un tipo de microprocesador especializado para tareas de procesado de señal. En las técnicas de procesado digital de señal hay determinado tipo de operaciones que son muy habituales. Por ejemplo realizar un filtro en digital supone multiplicar una colección de las últimas muestras de la señal por los respectivos valores (fijos) de la respuesta al impulso del filtro, y sumar todas esos productos parciales para obtener el resultado final. Los DSPs tienen una arquitectura interna que permite hacer esas cosas a velocidad máxima. Suelen tener varios buses de datos independientes ligados a un multiplicador y un acumulador (operación MAC); tienen instrucciones en su juego de instrucciones máquina que realizan todas esas funciones (que en un procesador convencional supondrían muchas instrucciones y por tanto aumentaría el tiempo de proceso); tienen modos de direccionamiento específicamente diseñados para procesado digital, como el direccionamiento *bit–reverse* que es útil al calcular FFTs para realizar procesado espectral (realizar ese direccionamiento en un procesador normal lleva muchos ciclos de máquina, y en los DSPs ese direccionamiento está resuelto por hardware y sólo gasta un ciclo de máquina).

Desde el punto de vista del uso, los DSP se pueden programar (como cualquier micro) en su propio lenguaje ensamblador aunque lo normal es programarlos en algún lenguaje de alto nivel (C y demás) y dejar que el compilador tome las decisiones de cómo implementarlo en el procesador. Posteriormente se puede medir el rendimiento del resultado y si en alguna parte se está gastando más tiempo de lo debido esas rutinas se pueden rediseñar directamente en lenguaje ensamblador y enlazar con el resto, para así asegurar unas mejores prestaciones.

El hecho de utilizar las FPGAs en lugar de procesadores digitales de señal en la industria ha sido impulsada por el hecho de que las FPGAs combinan las mejores ventajas de los ASICs y de los sistemas basados en procesadores. Ofrecen velocidades

temporizadas por *hardware* y fiabilidad, pero sin requerir altos volúmenes de recursos para compensar el gran gasto que genera un diseño personalizado de ASIC. Los circuitos integrados reprogramables tiene la misma capacidad de ajustarse que un software que se ejecuta en un sistema basado en procesador, pero no está limitado por el número de núcleos disponibles. A diferencia de los procesadores, los FPGAs llevan a cabo diferentes operaciones de forma paralela, por lo que éstas no necesitan competir por los mismos recursos. Cada tarea de procesos independientes se asigna a una sección dedicada del circuito integrado, y puede ejecutarse de manera autónoma sin ser afectada por otros bloques de lógica. Como resultado, el rendimiento de una parte de la aplicación no se ve afectado cuando se agregan otros procesos.

Hoy en día se utilizan con frecuencia los DSP con sus múltiples aplicaciones en procesado de señal, pero el DSP a partir de unas ciertas velocidades y una cierta complejidad resulta relativamente caro y demasiado lento, de hecho para circuitos de alta velocidad los DSP limitan mucho la velocidad de proceso.

Normalmente es más sencillo simular el funcionamiento de un sistema de forma software e incluso en las primeras pruebas puede ser mas rápido, eso si, si se dispone de una tarjeta de desarrollo con su hardware y software específico, el coste resulta ser más barato y el esfuerzo es menor realizando un diseño *hardware* en lugar de *software*. Las rutinas de control son más complejas pero a la hora de hacer funcionar al sistema y hacerle autónomo resulta más rentable.

A continuación voy a describir algunas de las ventajas de la utilización de las FPGAs en detrimento de los DSPs:

- **Rendimiento:** Aprovechando del paralelismo del hardware, las FPGAs poseen una potencia de cálculo muy superior a los procesadores digitales de señales (DSPs) rompiendo el paradigma de ejecución secuencial y realizando más operaciones en cada ciclo de reloj. La compañía Berkeley Design Technology, Inc. (BDTI), una destacada firma analista que realiza evaluaciones de referencia, lanzó evaluaciones mostrando cómo los FPGAs pueden entregar significativamente más potencia de procesamiento que una solución de DSP, en algunas aplicaciones y más económica [45]. El controlar entradas y salidas (E/S) a nivel de hardware

ofrece tiempos de respuesta más veloces y funcionalidad especializada que coincide con los requerimientos de una aplicación.

- **Tiempo en llegar al mercado:** La tecnología FPGA ofrece flexibilidad y capacidades de rápido desarrollo de prototipos para enfrentar los retos de que un producto llegue tarde al mercado. El diseñador puede probar una idea o un concepto y verificarlo en hardware sin tener que pasar por el largo proceso de fabricación por el que pasa un diseño personalizado de ASIC. Posteriormente podrá implementar cambios y realizar iteraciones de un diseño FPGA en cuestión de horas en vez de semanas. También existe *hardware* comercial listo para usarse con diferentes tipos de E/S ya conectados a un chip FPGA programable por el usuario. El aumento en disponibilidad de herramientas de software de alto nivel disminuye la curva de aprendizaje con niveles de abstracción. Estas herramientas frecuentemente incluyen importantes núcleos IP (funciones pre–diseñadas) para control avanzado y procesamiento de señales.

- **Precio:** El precio de la ingeniería no recurrente de un diseño personalizado ASIC excede considerablemente al de las soluciones *hardware* basadas en FPGA. La fuerte inversión inicial de los ASICs es fácilmente justificable para los fabricantes de equipos originales que embarcan miles de chips por año, pero muchos usuarios necesitan la funcionalidad de un *hardware* personalizado para decenas o cientos de sistemas en desarrollo. La misma naturaleza programable del circuito integrado implica que no hay precio de fabricación o largo plazos de ejecución de ensamblado. Los requerimientos de un sistema van cambiando con el tiempo, y el precio de cambiar incrementalmente los diseños FPGA es insignificante al compararlo con el precio de implementar cambios en un ASIC antes de su lanzamiento comercial.

- **Fiabilidad:** Mientras que las herramientas de software ofrecen un entorno de programación, los circuitos de una FPGA son una implementación segura de la ejecución de un programa. Los sistemas basados en procesadores frecuentemente implican varios niveles de abstracción para ayudar a programar las tareas y compartir los recursos

entre procesos múltiples. El software a nivel *driver* se encarga de administrar los recursos *hardware* y el sistema operativo administra la memoria y el ancho de banda del procesador. El núcleo de un procesador sólo puede ejecutar una instrucción a la vez, y los sistemas basados en procesadores están siempre en riesgo de que sus tareas se obstruyan entre sí. Las FPGAs, que no necesitan sistemas operativos, minimizan los problemas de fiabilidad con ejecución paralela y *hardware* preciso dedicado a cada tarea.

- **Mantenimiento a largo plazo:** Como se mencionó anteriormente, los circuitos integrados FPGA son actualizables en campo y no requieren el tiempo y el precio que implica rediseñar un ASIC. Los protocolos de comunicación digital por ejemplo, tienen especificaciones que podrían cambiar con el tiempo, y las interfaces basadas en ASICs podrían causar retos de mantenimiento y habilidad de actualización. Los chips FPGA, al ser reconfigurables, son capaces de mantenerse actuales con modificaciones que pudieran ser necesarias en un futuro. Mientras el producto o sistema se va desarrollando, el diseñador puede implementarle mejoras funcionales sin la necesidad de invertir tiempo rediseñando el *hardware* o modificando el diseño de la tarjeta.

5 Cálculo de la convolución en FPGAs disponible en la bibliografía

RESUMEN

Este capítulo se dedica al estudio y comentario de los artículos disponibles en la bibliografía sobre el tema de la convolución en los dispositivos FPGAs. Posteriormente se comentan los estudios realizados sobre FPGAs sobre aritmética, en particular se comienza sobre los que utilizan la aritmética carry-save. A continuación se estudian los que tratan la aritmética en general, básicamente sumadores y multiplicadores. Posteriormente se detallan las investigaciones realizadas sobre FPGAs que contienen multiplicadores empotrados y los que realizan su estudio sobre bloques DSPs empotrados en la FPGA, para terminar con los que están dedicados solamente a la arquitectura multiplica–acumula.

5.1. Introducción

La operación de convolución es una de las operaciones básicas en el procesamiento digital de señales, ya que tendrá sus aplicaciones en el filtrado, transformadas (FFT), correlación, etc. Está basada en la operación de multiplicación y acumulación (operación MAC). Para reducir el costo computacional de las multiplicaciones implicadas en estas aplicaciones normalmente se implementa un multiplicador o se usa aritmética distribuida. No obstante, en otras aplicaciones la multiplicación no se puede evitar (filtros adaptativos, correlación, redes neuronales,...). Esto ha obligado a los fabricantes de dispositivos FPGA a usar multiplicadores empotrados para llevar a cabo este tipo de operaciones de forma eficiente.

La convolución implica muchas multiplicaciones y acumulaciones. Si una FPGA dispone de pocos multiplicadores, es frecuente hacer uso de ellos de forma iterativa para implementar esta función. Muchos autores han implementado diferentes diseños para llevar a cabo la operación de convolución. Los actuales dispositivos FPGA incluyen una lógica de acarreo que permite la implementación de sumadores con acarreo propagado rápido, como quedó expuesto en el capítulo 4, para el caso de las FPGA Spartan 3 y

Spartan 6 de Xilinx. En esta tesis se propone el uso de sumadores *carry–save* como una alternativa eficiente para reducir el tiempo de cálculo de la convolución. Está basado en una asignación a bajo nivel de los propios recursos del *slice* de la FPGA, que ofrece un rendimiento optimizado en comparación con la asignación automática. Por otra parte, Actualmente existe un creciente interés de la comunidad científica en el diseño de nuevos algoritmos que aprovechen la arquitectura interna de las FPGA.

En este capítulo se detallan, los estudios realizados por diversos autores sobre el tema de la convolución en dispositivos FPGA, sobre los que utilizan aritmética *carry–save* y sumadores o multiplicadores en general, para posteriormente pasar a los que tienen en cuenta los multiplicadores y bloques DSP empotrados dentro de la FPGA. Finalmente se realiza el estudio sobre algunos artículos que versan sobre aplicaciones de la convolución, principalmente el filtrado de señales digitales (filtros FIR) y sobre los que se centran sobre la arquitectura MAC, que es la operación que se debe optimizar sobre cualquier dispositivo electrónico, en el caso de esta tesis, las FPGAs, ya que el multiplicador–acumulador constituye el "cuello de botella", para la operación de convolución.

5.2. Estado del arte sobre la convolución en FPGA

Existen muchos estudios actualmente sobre la convolución. Citando los más actuales, los autores Hashemi & Eshghi en [46] proponen una arquitectura para la convolución donde los registros de entrada son registros de tipo de entrada–serie/salida–paralelo, mostrando los resultados sobre una FPGA Spartan–3E de Xilinx. En este artículo muestran los datos obtenidos pero solamente para datos de entrada de hasta 8 bits, sin comentar la arquitectura interna de la FPGA, que dispone de multiplicadores de 18x18 bits. Se basan solamente en la forma de introducir los datos en la FPGA lo cual hemos estudiado y publicado en el artículo [47] donde se explica una arquitectura basada en buffers circulares. Estos autores justifican con estos datos que el aumento de velocidad es mucho mayor que el incremento en área, lo cual es lógico para datos de un pequeño número de bits. El tipo de sumadores que emplea son los típicos *ripple carry adders* y los multiplicadores empotrados de que dispone la Spartan–3.

Por otro lado Mohammad & Agaian en [48] proponen un circuito de convolución 4x4, lo cual es demasiado pequeño comparado con los dispositivos FPGA actuales que disponen de multiplicadores, con resultado hasta 48 bits, lo cual hace que este circuito quede algo

obsoleto, para nuestros objetivos. Este circuito puede expandirse para construir la convolución de cualquier número de bits, pero no muestran resultados. Por otro lado, no mencionan sobre qué FPGA han implementado el circuito aunque indican que han utilizado la herramienta ISE de Xilinx. Utilizan el lenguaje de descripción hardware Verilog. Sí justifican el hecho de no utilizar la convolución vía FFT, ya que este algoritmo utiliza muchos recursos hardware, porque el resultado de la FFT hay que almacenarlo en cada LUT de la FPGA. El método de la FFT, consiste en realizar la transformada de Fourier de las dos señales de entrada, con lo que se tienen las dos señales en el dominio de la frecuencia. La convolución en el dominio del tiempo se traduce en un producto en el dominio de la frecuencia, con lo que posteriormente hay que realizar la transformada inversa de Fourier. Aunque el algoritmo de la transformada rápida de Fourier de Cooley & Tukey [3] fue muy extendido en los procesadores digitales de señal (DSPs), para los dispositivos FPGA ha dejado de ser útil en términos de área consumida en dicho dispositivo. Los DSP disponen de un modo de direccionamiento de *bit–reverse* propio para realizar este algoritmo de la FFT.

Los autores Wang et al. en [49] proponen un algoritmo de convolución de descomposición de la señal de entrada en paralelo, y la partición de los coeficientes del filtro en su partes par e impar sucesivamente, con lo cual proponen realizar cada una de las sub–convoluciones en distintas FPGAs, en su ejemplo lo aplican a cinco FPGAs Virtex4sx55 de Xilinx. La aplicación de esta propuesta es sobre un simulador de eco de radar. La solución es ventajosa porque aumenta mucho la velocidad ya que los datos de entrada en el eco de radar necesita especificaciones de tiempo real, pero resulta muy costosa en hardware; es decir, no es de bajo coste utilizar 5 FPGAs Virtex–4. Por otra parte, no especifican para nada la arquitectura de la FPGA, simplemente la simulan.

Los doctores Ernest Jamro y Kazimierz Wiatr de la Universidad de Ciencia y Tecnología de Cracovia (Polonia), han creado la herramienta software *AuToCon* (*Automated Tool for generating Convolution in FPGA*). En varios artículos los autores realizan la implementación en FPGA de la operación de convolución. En particular en [50] proponen la implementación de la suma como una parte de la convolución. Describen tres tipos de multiplicadores para el cálculo de la convolución: la multiplicación sin multiplicadores (*multiplierless multiplication*) donde se utiliza el método de desplazamiento del multiplicando y su posterior adición; la multiplicación basada en tablas

de búsqueda (*lookup table*, LUT), que es la que se utiliza en los resultados de esta tesis; y una tercera opción, que consiste en la utilización de aritmética distribuida. En este artículo defienden el uso de sumadores RCA porque las FPGAs disponen de una lógica rápida de acarreo. En el capítulo siguiente de la tesis se justificará el uso de sumadores *carry–save* cuando el número de operaciones multiplica–acumula es muy grande, como es el caso de la convolución. Proponen el uso de árboles de sumadores basados en LUT.

Otro estudio interesante es una nota de aplicación de Thomas Oelsner de la compañía QuickLogic Europe [51]. En esta nota de aplicación se justifica el hecho de la utilización de la FPGAs para operaciones de procesamiento digital de señales en detrimento de los procesadores digitales de señal (DSPs). Obviamente la implementación se realiza sobre las FPGAs de esta compañía, describiendo su arquitectura. Pero lo más interesante de esta nota es el algoritmo de la convolución en VHDL y en Verilog, aunque nombra el teorema de la convolución: *La transforma de Fourier de la convolución de dos señales es el producto de sus transformadas de Fourier individuales*. Ya comenté anteriormente que esta forma de calcular la convolución está prácticamente descartada cuando se implementa en FPGA, debido al aumento de hardware consumido para almacenar la transformada de cada una de las dos señales de entrada. En esta nota se describen los recursos utilizados de la FPGA y los tiempos que tarda en realizar la operación MAC.

A continuación, se describen dos artículos que utilizan la convolución para una aplicación concreta como es el filtrado de imágenes. En concreto unas investigaciones del Instituto de Microelectrónica de Sevilla, Centro Nacional de Microelectrónica (IMSE-CNM-CSIC) los investigadores Garcés Socarrás et al. realizan un estudio [52] donde exponen el diseño de bloques de convolución para procesado de imágenes con FPGA. Los bloques han sido desarrollados empleando un flujo de diseño basado en modelos que se apoya en el entorno MATLAB/Simulink y la herramienta *System Generator* de Xilinx. Su implementación física se ha llevado a cabo sobre una placa de desarrollo Spartan–3A DSP 1800 de Xilinx. Muchas de las etapas de un sistema de procesado lineal de imágenes se basan en la operación matemática de convolución de la imagen, representada por una matriz de dos dimensiones, con otra matriz (de 3´3 ó 5´5 píxeles en la mayoría de los casos) denominada kernel de convolución. La definición matemática del algoritmo de convolución en dos dimensiones permite explotar las posibilidades de paralelismo de ejecución y el diseño de filtros específicos para kernels predefinidos.

Por otro lado, existe otra nota de aplicación, en este caso de Xilinx, en la que se describe la operación de convolución en dos dimensiones, donde parametrizan los valores de las entradas descomponiendo el filtro FIR en vertical y horizontal. Utilizan su herramienta *System Generator* para realizar los cálculos, obteniendo unos resultados para las FPGAs de Xilinx Virtex-II Pro y Spartan 3, indicando la frecuencia obtenida y los recursos empleados (número de slices, bloques RAM y multiplicadores o bloques DSP).

Termina este apartado comentando dos artículos que implementan la convolución en FPGA pero utilizando una matemática algo peculiar, la matemática *Vedic*. Esta matemática se explica en un libro publicado en 1965 escrito por Bharati Krishna Tirthaji [53]. Contiene una lista de 16 técnicas de cálculo mental que están basadas en el antiquísimo Vedas. Una de estas técnicas, llamadas *sutras*, es el algoritmo de multiplicación *Urdhva Tuiryagbhyam*, que es el que utilizan estos dos artículos Kulkarni [54] y Haveliya [55]. Justifican el uso de estos algoritmos de la matemática *Vedic* porque es fácil de comprender, aplicar y recordar, por lo que es útil para programadores y en el ámbito científico. En el primero de ellos ponen como ejemplo que las dos señales son de 4 bits, x[n] = {a_3 a_2 a_1 a_0} y h[n] = { b_3 b_2 b_1 b_0}; al realizar la convolución, los términos y_0 e y_6 se obtienen directamente tras realizar la multiplicación, los términos y_1 e y_5 ya tienen que realizar una suma para lo cual proponen un sumador de acarreo anticipado (*Carry Look Ahead*, CLA) y para los términos intermedios y_2, y_3 y_4 utilizan sumadores *carry–save* seguidos de un sumador *ripple carry* final. Los resultados implementados en una Spartan 3E de Xilinx, dan los mejores resultados de la combinación CSA–RCA

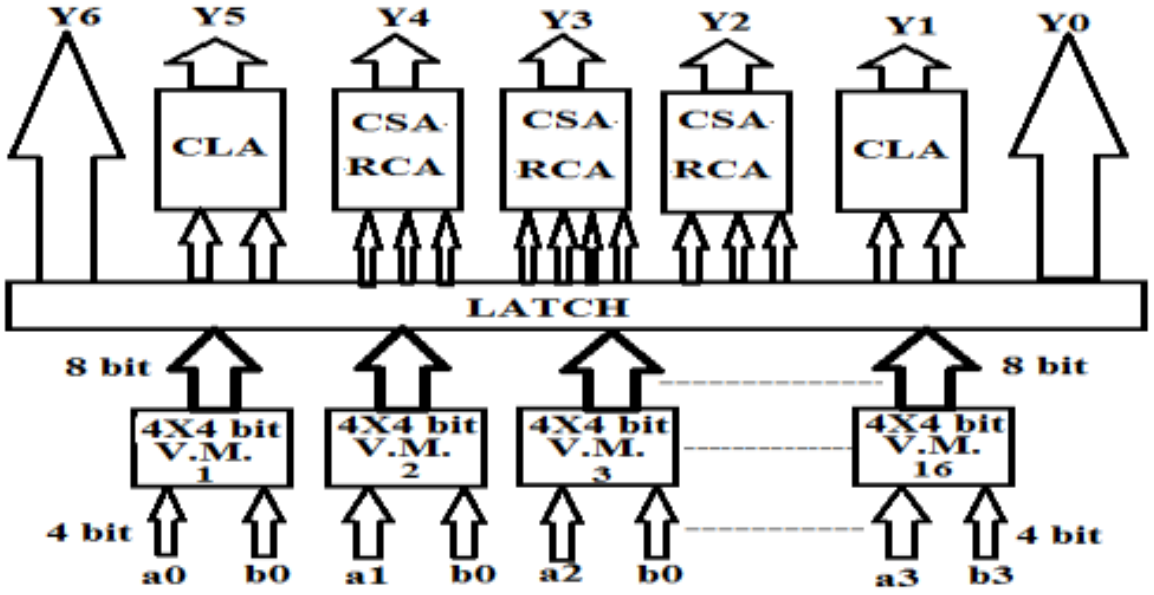


Figura 5-1 Diagrama de bloques propuesto para la convolución.

Este diseño lo han realizado con el lenguaje en VHDL e implementado en una Spartan 3E de Xilinx utilizando la herramienta Xilinx ISE *design suite* 9.2i. En el segundo artículo [55] utilizan el mismo multiplicador de 4 x 4 pero a partir del método *overlap–add* (OLA), con el que reducen un multiplicador NxN en estructuras de 4x4. En este caso la convolución la han implementado con el lenguaje de programación hardware VHDL sobre una FPGA Virtex-6 utilizando la herramienta software Xilinx ISE *design suite* 12.1.

5.3. Aritmética carry–save en FPGA

En este apartado se estudian algunos de los artículos disponibles en el bibliografía que utilizan aritmética *carry–save* implementada sobre dispositivos FPGA, por ser la utilizada en los cálculos experimentales que se exponen en el capítulo siguiente de esta tesis. El uso de la aritmética *carry–save* en FPGA en los últimos años ha sido descartada por varios motivos. En primer lugar, por el pequeño tamaño de los operandos utilizados en las aplicaciones típicas para FPGAs; en segundo lugar el buen equilibrio de los sumadores de acarreo propagado para pequeño tamaño de bits utilizando la lógica de acarreo de que disponen este tipo de dispositivos; y finalmente el excesivo consumo de área de dispositivo que utilizan las herramientas software de diseño sobre las unidades de aritmética *carry–save*.

La representación redundante de datos es utilizada para reducir el tiempo empleado en la suma limitando la longitud de la propagación del acarreo a uno o dos dígitos como máximo. Por tanto, el resultado de la suma no depende de la longitud de los operandos. Las representaciones más usuales son *carry–save* (CS) y *signed–digit* (SD). La idea básica de un sumador *carry–save* es la suma de tres números usando un *array* de sumadores completos de 1 bit pero sin propagación del acarreo. El resultado es un número redundante en representación CS compuesto por una palabra de suma (S) y una palabra de acarreo (C). Por tanto la suma de tres números x, y, z de n bits es representada por dos números C y S.

$$x + y + z = C + S$$

Los números C y S de n bits son obtenidos sin propagación de acarreo consumiendo el tiempo de un único sumador completo. Esta representación realiza una reducción de tres números binarios a dos números binarios por lo que se llama sumadores [3:2].

Un CSA es capaz de realizar la suma de tres operandos convencionales cualesquiera o bien la suma de uno convencional y otro operando en formato CS que es el caso necesario para realizar la operación de acumulación.

En primer lugar los autores Ortiz et al. en [36] justifican con datos el uso de aritmética redundante con un coste en recursos *hardware* próximo al que consumen los sumadores con acarreo propagado. Aunque en los dispositivos FPGA no es muy común utilizar sumadores *carry–sa*ve, debido a que en principio, utilizan el doble de hardware que los sumadores con propagación de acarreo, en este artículo se demuestra que para sumas con elevado número de bits (mayor de 16), la suma *carry–save* llega a ser ventajosa, ya que son claramente más rápidos utilizando aproximadamente el mismo hardware. En concreto realizan el estudio sobre la arquitectura de las FPGAs de Xilinx que disponen de LUT de cuatro entradas y lógica de acarreo especializada, tales como Virtex 2, 4 o Spartan 2, 3.

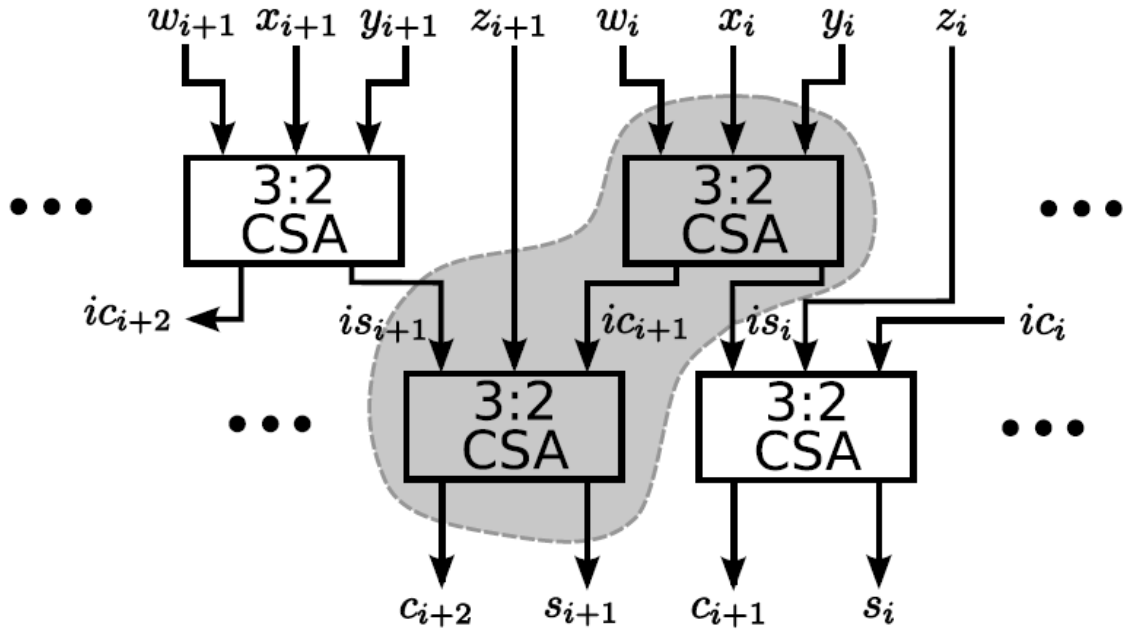


Figura 5-2 Implementación de compresores 4:2 utilizando contadores 3:2

Si queremos sumar dos números en representación CS, se requiere una reducción 4 a 2 (sumador [4:2]) que puede ser implementado por dos CSA como se indica en la figura 5-2. En este caso, el tiempo para calcular la suma de dos números CS de n bits es el utilizado por dos sumadores completos. El mapeo de un compresor 4:2 en un *slice* de estas FPGAs se indica en la figura 5-3, donde está incluida la parte sombreada de la figura 5-2.

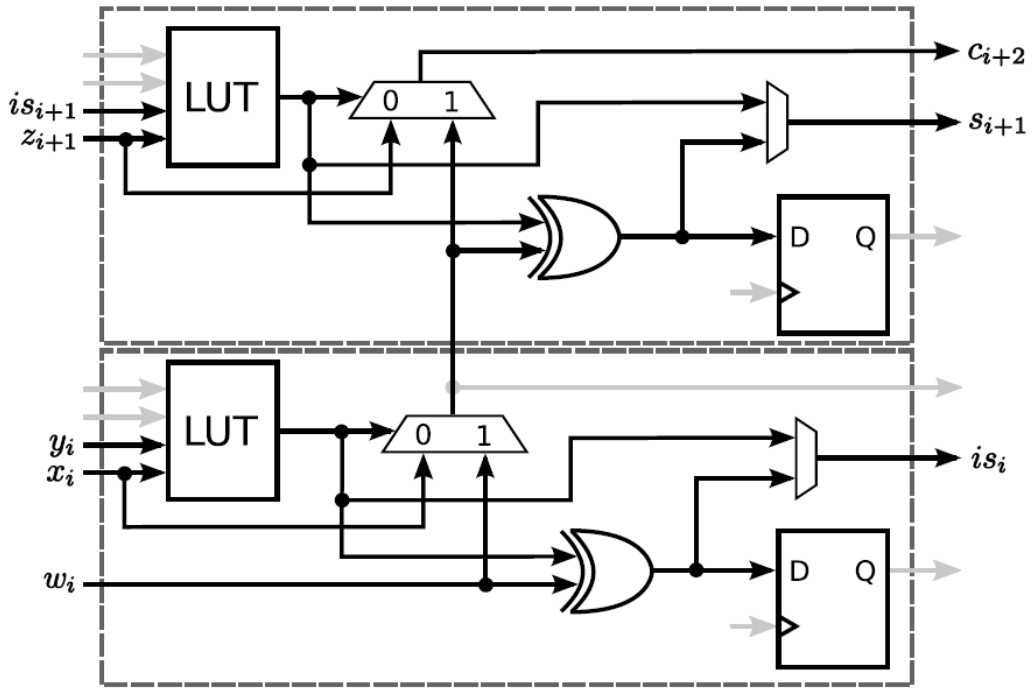


Figura 5-3 Mapeo de un compresor 4:2 en un slice

En un intento de utilizar la lógica de acarreo disponible en la FPGA, manteniendo el retardo máximo de un sumador independientemente de la longitud de palabra del operando, los autores Beuchat y Muller [11] presentan una solución utilizando una representación *carry–save* de alto índice. Esta representación se introdujo incialmente para reducir el número de conexiones y registros necesarios para almacenar el valor suma en *radix–r* y la palabra de acarreo necesita un solo bit por dígito en *radix–r*. En este artículo se descartaba la solución de utilizar aritmética CSA debido a que se dejaba sin utilizar el semi–-slice superior para sus aplicaciones, pero para otras aplicaciones, tal y como se describía en el artículo anterior [36] este se utilizaba para la realización de un compresor 4:2.

Por otro lado, Verma et al. en [56] proponen una agrupación de sumas en punto flotante en las FPGA de Altera Stratix III, utilizando aritmética *carry–save*. La propuesta consiste en un método de suma que mejora los resultados del compilador de Altera en punto flotante, utilizando aritmética *carry–save* para sumas multi–operando, denormalizando los valores de entrada en punto flotante en paralelo.

Otro ejemplo del uso de la aritmética *carry–save* es el propuesto por Gutiérrez et al. [57] donde explican un método para la multiplicación con coeficientes constantes multiplexadas en el tiempo. Describen la arquitectura de los slices de la FPGA Virtex-5 de

Xilinx, con la posibilidad de configurar los *slices* de dicha FPGA para realizar la operación en complemento a dos y un doble multiplexor, para seleccionar entre tres entradas diferentes, para realizar la operación *carry–save* o para realizar la suma final CPA. La aplicación de esta propuesta la realizan sobre dos ejemplos: la implementación de la transformada rápida de Fourier de n puntos, y para la transformada discreta del coseno en 2–D. Los resultados experimentales los implementan sobre la FPGA Virtex–5 de Xilinx. Los resultados muestran que el método propuesto mejora el rendimiento hasta en un 50% y reduce el área en muchos casos en comparación con los sumadores de acarreo propagado.

La aritmética *carry–save*, que ha sido olvidada en la pasada década, está siendo justificada en los últimos años, debido a la utilidad en el operaciones del procesamiento digital de señales, cuando el número de operandos es elevado. Por ejemplo en otros estudios, Erle et al. en [58] y Gustafsson en [59], diseñan bloques multiplicadores utilizando sumadores *carry–save* y la multiplicación en punto flotante utilizando este tipo de aritmética.

5.4. Estudios sobre sumadores y multiplicadores en FPGA

Se describen a continuación algunos artículos que estudian los sumadores, multiplicadores y demás circuitos artitméticos que no han sido comentados anteriormente, bien por que no tratan directamente la convolución, o porque no utilizan directamente la aritmética *carry–save*. En particular en este apartado comenzaré por comentar un artículo que utiliza aritmética redundante para sumadores multioperando. Hormigo et al. en [60] defienden el uso de aritmética redundante en dispositivos FPGA, utilizando árboles de compresores *carry–save* en estos dispositivos que optimizan la lógica de acarreo disponible en cualquier familia de FPGA y para distintos fabricantes (Altera y Xilinx). En este estudio realizan varios árboles de compresores de diferentes tamaños y los implementan en FPGA de Xilinx (Spartan 3A, Virtex 4 y Virtex 5), proporcionan datos del área utilizada y de los retardos obtenidos. Comparan sus resultados con los de otros investigadores que utilizan contadores de propósito general (GPC). Como resultado se obtiene una implementación eficiente de estos árboles de compresores en FPGA, en términos de área consumida y velocidad utilizando la especializada lógica de acarreo de estos dispositivos.

Por otro lado, Hoe et al. [61] estudian tres tipos de sumadores *carry–tree* como son *Kogge–Stone adder*, *sparse Kogge–Stone adder* y *Spanning tree carry lookahead adder*

(CLA). Los comparan con el clásico *ripple carry adder* (RCA) y llegan a la conclusión de que no son demasiado útiles para moderados anchos de banda. Han utilizado el software de Xilinx ISE 12.1 para sintetizar los diseños en la Spartan 3E. Sobre esto habría que comentar que en las aplicaciones actuales del procesamiento digital de señales son muchos los operandos con los que se trabaja, para lo cual es útil la aritmética redundante como se justificó en Hormigo et al. [60].

En el estudio realizado por Bhattacharjee et al. [62] se comparan diversos circuitos aritméticos en la FPGA Virtex–5 de Xilinx, en términos de área consumida (*slices*), retardo y consumo de potencia. En concreto, los sumadores que comparan son *ripple–carry adder* (RCA), *carry–select adder*, *carry–lookahead adder* (CLA), *carry–skip adder* y *sign–magnitude adder* (SMA), que ya se describieron en el capítulo 3 de esta tesis. Por otro lado los multiplicadores comparados son los siguientes: multiplicador básico, multiplicador *carry–save*, multiplicador *carry–ripple* y multiplicador de Booth para operandos con signo. Como conclusión obtienen que los diseños con menor número de *slices* utilizados consumen menos energía, y que las implementaciones con retardos más pequeños también consumen menos energía. Realizan la implementación para circuitos con aritmética en punto fijo de 8, 16, 32 y 64 bits.

5.5. Multiplicadores empotrados y bloques DSP en FPGA:

En los últimos años, la mayoría de los fabricantes de FPGA han incluido en sus dispositivos diversos bloques empotrados, tales como multiplicadores empotrados y bloques DSP para optimizar las operaciones básicas que se emplean en el procesamiento digital de señales. Para el caso de Xilinx, ya fueron descritos detalladamente en el capítulo 4 por ser los que se utilizan en los resultados de la tesis. Otros fabricantes también han incluido bloques DSP como es el caso de Altera en sus dispositivos Stratix II y Stratix II GX, por ejemplo.

Voy a centrarme en dos grupos de autores. Por un lado Shuli Gao, Dhamin Al–Khalili y Noureddine Chabini, *Department of Electrical and Computer Engineering, Royal Military College of Canada, Kingston, Ontario, Canada*; investigadores que han publicado diversos artículos en los cuales utilizan los bloques DSP de las dos compañías Xilinx y Altera. Por empezar con el más actual, en [63] presentan una optimización de multiplicadores de gran tamaño en complemento a 2, utilizando los bloques

multiplicadores de la Spartan 3 de Xilinx. Esta realización está basada en el algoritmo de Baugh–Wooley [64] utilizan la plataforma de diseño ISE 8.1 de Xilinx y la comparan con otras tres aproximaciones: aproximación convencional con extensión de signo, utilizando Xilinx's IP–Core Generator y una aproximación basada en signo–magnitud. En este artículo hacen referencia a otro de los mismos autores [65] en donde no hacían la descripción tan detallada del algoritmo.

Los mismos autores publicaron otro artículo [66] donde utilizaron los bloques DSP de las FPGA de Altera, que pueden ser configurados como multiplicadores de 9×9, 18×18, or 36×36 bits. Esta flexibilidad del funcionamiento multi–granular de pequeños multiplicadores empotrados la utilizan para optimizar funciones de gran tamaño en número de bits, tales como la multiplicación de un gran tamaño de bits. En concreto en este artículo desarrollan multiplicadores con signo de 256x256 bits a partir de los bloques DSP de 18x18 y 36x36 bits de que dispone la Stratix II de Altera.

A continuación, se describen los estudios de un grupo de autores que han publicado varios artículos. Unos se refieren a los bloques DSP de que disponen la mayoría de las FPGAs actuales y los otros artículos definen los compresores y unos bloques aritméticos que denominan FPCA (*Field Programmable Counter Array*). Estos autores son Hadi Parandeh–Afshar, Paolo Ienne, Philip Brisk y Ajay Kumar Verma.

En uno de sus artículos [67] abordan la discrepancia entre las FPGAs y los ASIC proponiendo una nueva red reconfigurable para acelerar los componentes aritméticos de un circuito. A estos módulos para aritmética le llaman FPCA (*Field Programmable Counter Array*) y pueden integrarse en el mismo circuito FPGA. El mejor circuito para calcular sumas de varios operandos es un compresor construido a partir de sumadores *carry–save* en lugar de sumadores con acarreo propagado, que solamente será necesario para calcular la suma final (S+C). Existen muchas formas de contruir compresores, una de ellas es a partir de contadores m:n (m es el número de entradas y n el número de salidas), que consiste en contar el número de "unos" de los bits de entrada y n es el valor en binario de dicho número. El número de bits necesario a la salida es $n = \log_2(m + 1)$. Los autores proponen estos bloques FPCA que son similares a una FPGA donde los CLB se sustituyen por contadores m:n. Los resultados experimentales en este artículo demuestran que los circuitos FPCA son más rápidos para sumas relativamente pequeñas y proporcionan los resultados en la Virtex 4 de Xilinx y Stratix–II de Altera. Comentan que se obtienen

mejores resultados en la FPGA de Altera que utilizando los bloques DSP48E de la Virtex–4, realizando ejemplos de filtros FIR (operación de convolución).

Por otro lado, en otros artículos [68] y [69] demuestran que los árboles de compresores pueden ser mapeados eficientemente en las FPGA actuales proporcionando resultados en la FPGA Stratix-II de Altera. En FPGA más actuales se han introducido las LUT de 6 entradas, que pueden ser configuradas como varias LUT de menores entradas. Por ejemplo las LUT de 6 entradas de Altera pueden ser configuradas como dos LUT de 3 entradas que permiten el cálculo de los registros *carry* y *save* en aritmética CSA, lo que les permite actuar como compresores 3:2 y junto con la lógica de acarreo, el bloque lógico puede ser configurado como un sumador ternario CPA. Otra alternativa es integrar cores IP en hardware, como los bloques DSP/MAC, dentro de la FPGA. Pero en otro artículo los investigadores Kuon y Rose [70] argumentan que no ofrecen muchas ventajas por dos razones: la primera de ellas es que al ser fija la posición de los bloques DSP el coste del ruteo es alto, y la segunda es que presentan desajustes en el ancho de banda, como por ejemplo emplear multiplicadores 18x18 para realizar multiplicaciones de 9x9. Por tanto, para las operaciones cuyo ancho de bits no coincide con la del bloque DSP, es preferible la flexibilidad de las LUT. Para el mapeo de los compresores proponen el uso de los GPC (*generalized parallel counters*) introducidos en 1977 por Stenzel et al. [71].

Los mismos autores en [72] demuestran que la lógica de acarreo de que disponen las FPGAs actuales puede ser utilizada para aritmética *carry–save*, y no sólo para la suma de propagación de acarreo. Definen un método para detener el acarreo no más de dos bits, utilizando además de la lógica de acarreo de que dispone la FPGA, la programación eficiente de las LUT. Construyen los árboles de compresores, de manera similar al artículo anterior, con GPC, pero combinando las LUT con la lógica de acarreo, en lugar de sólo con las LUT. Este diseño lo llevan a cabo en el módulo lógico adaptativo (*adaptative logic module*, ALM) que constituye la celda lógica de las series FPGA Stratix II y IV de Altera. Con esta técnica consiguen similares retardos que los compresores de los artículos anteriores pero mejorando el área, es decir reduciendo el número de ALM.

Parandeh–Afshar et al. en otros dos artículos [73] y [74] proponen un nuevo bloque DSP para mejorar el rendimiento de la aritmética en las FPGAs actuales. Realizan la comparación de su nuevo bloque DSP con los bloques de que dispone la Stratix II de Altera, haciéndolos más flexibles.Acelera la operación de adición con muchos sumandos y

la operación MAC. Los cálculos experimentales los implementan sobre la citada FPGA, midiendo los retardos de los multiplicadores de diferentes tamaños de bits, con tres tipos de implementaciones: utilizando los bloques DSP de la Stratix II, los FPCT (*field programmable compressor tree*) y su nuevo bloque DSP propuesto. Para la adición de múltiples entradas, operación que no está soportada en los bloques DSP, el bloque DSP propuesto es más de 50% más rápido en promedio en comparación con el software de la FPGA.

Los cuatro autores publicaron un artículo [75] en IEEE *Transactions on Very Large Scale of Integration System* (2010), donde mejoran el rendimiento de las FPGA para la aritmética *carry-save* que puede considerarse un compendio de varios artículos. En él introducen su FPCA que es un acelerador para aritmética *carry–save* integrado en la FPGA como alternativa a los bloques DSP. En sus resultados experimentales demuestran que con su propuesta aceleran un mayor número de aplicaciones DSP comparado con los bloques DSP y mejoran su rendimiento, área y consumo de energía comparado con el software de las FPGA. Realizan comparaciones utilizando sus FPCA con diferentes tamaños de contadores, con los bloques DSP, árboles de compresores utilizando los sumadores ternarios de la FPGA, los GPC. Obtienen como conclusión la mejora tanto en área como en retardo de sus FPCA para distintas aplicaciones: DCT, FIR, multiplicadores de diferentes tamaños, etc.

En otra de sus investigaciones [76] estudian la diferencia de rendimiento entre los multiplicadores empotrados y los multiplicadores software en las FPGA. Para ello desrrollan una herramienta de generación de multiplicadores realizada en C++, donde implementan dos técnicas de multiplicación convencional. Comparan sus multiplicadores con los generados por la herramienta de Altera para distinto número de bits. Esta herramienta toma los datos de entrada con signo o sin signo, con distintos números de bits en la entrada y genera el *netlist* del multiplicador en Verilog a bajo nivel. Realizan las pruebas sobre la FPGA Stratix III de Altera y la comparan con la herramienta Altera Quartus–II Megawizard, llegando a la conclusión de que sus resultados son un 27 % más rápidos.

Voy a terminar este apartado referenciando dos artículos de Kwon et al. [77] y [78] donde diseñan una unidad MAC de 16 x 16 bit utilizando compresores 5:3 y 5:2. Lo he dejado para el final de este apartado porque la implementación no es realizada sobre

FPGA, que es el circuito que interesa en esta tesis. A partir de los contadores 3:2 describen la realización de los compresores 4:2, compresores 5:3 y compresores 5:2, a nivel de puerta XOR y multiplexores. Como ejemplo de diseño de estos compresores exponen una unidad MAC de 16 bit x 16 bit utilizando el compresor 5:3 definido. Con los datos experimentales concluyen que su nuevo compresor 5:3 mejora la velocidad un 14.3% cuando utilizan puertas XOR, y para circuitos CMOS dinámicos mejora la velocidad un 11.7% y un 8.1% menos en consumo de potencia.

5.6. Aplicaciones de la convolución en FPGA

Los artículos que se mencionan a continuación tienen como tema principal de aplicación los filtros digitales, y más en concreto los filtros de respuesta finita al impulso (filtros FIR) por ser los más aplicados y estudiados en el procesamiento digital de señales. La expresión matemática de la convolución era:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Un filtro FIR es la convolución de la señal de entrada con los coeficientes del filtro C_i. La salida de un filtro de orden (longitud) L de una señal de entrada x[n] viene dada por una versión finita de la suma de convolución de esta señal de entrada con los coeficientes del filtro (que son constantes).:

$$y[n] = \sum_{k=0}^{L} x[k] \cdot Ci[k]$$

Básicamente en la bibliografía actual existen dos algoritmos para la realización de estos filtros: utilizando un algoritmo de multiplicación–acumulación, o utilizando aritmética distribuída. En el primero de ellos los datos de salida se calculan retardando, sumando y multiplicando los valores de las muestras, y en el segundo se precalcula el valor de los coeficientes lo que permite ahorrar gran número de recursos internos en la FPGA. Los autores Rawski et al. en [79] basan la realización de filtros FIR utilizando aritmética distribuída. En este tipo de aritmética se contempla que, en lugar de la suma de productos clásica, la suma de productos de un bit concreto de una muestra de entrada se calcula en un solo paso sobre todos los coeficientes del filtro.

La aritmética distribuída es un método de calcular la suma de productos. El uso de la aritmética distribuída en el procesamiento digital de señales se justifica por su potencia de

cálculo. Su principal ventaja es su eficiencia de mecanización y la disminución de recursos internos, frente a la utilización del algoritmo MAC que permite frecuencias de funcionamiento más elevadas para un reducido número de bits. Por otro lado, aunque un código de un filtro con aritmética distrbuída es más largo y complejo, es más uniforme y el proceso de síntesis es más controlado por el diseñador en todo momento, mientras que con el algoritmo MAC se permite más libertad al sintetizador y su eficacia depende de la capacidad de esta herramienta de síntesis de implementar la operación de multiplicación, que consume un gran número de CLBs y es más dependiente del número de bits de los operandos.

Para diseños con mayor número de coeficientes, aumenta la diferencia entre el número de recursos utilizados entre ambos métodos. En un filtro MAC, el hecho de aumentar el número de coeficientes del filtro aumenta considerablemente el número de multiplicaciones y por tanto disminuye la frecuencia de funcionamiento del filtro; sin embargo, en un filtro con aritmética distribuída se produce un aumento del número de LUT menos significativo, además la frecuencia de funcionamiento se ve menos afectada.

En muchas aplicaciones del procesamiento digital de señales no se necesita una multiplicación de propósito general, por ejemplo, en el caso de la implementación de filtros, si los coeficientes del filtro son constantes en el tiempo (sistema LTI) los valores de la señal de la ecuación anterior son multiplicados por una constante. Teniendo en cuenta el hecho de que la señal de entrada es un número binario que podemos expresarlo de la forma siguiente:

$$x[n] = \sum_{b=0}^{B-1} x_b[n] \cdot 2^b \qquad \text{donde } x_b[n] \in [0,1]$$

la suma de convolución puede ser descrita así:

$$y[n] = \sum_{b=0}^{B-1} 2^b \cdot \sum_{k=0}^{L} x_b[k] \cdot c[k] = \sum_{b=0}^{B-1} 2^b \cdot \sum_{k=0}^{L} f(x_b[k], c[k])$$

El método de implementación más usual de la función $f(x_b[k],c[k])$ es mediante un módulo combinacional de L entradas donde la función f se presenta como una tabla de verdad que incluye todas las posibles combinaciones lineales de los coeficientes del filtro y los bits de la señal de entrada muestreada. En este artículo, para los datos experimentales han tomado varios ejemplos de filtros de diferente orden (L). Ellos proponen en este

estudio una implementación paralela de esta aritmética distribuída llegando a la conclusión que se obtiene una mejor velocidad y una mejor utilización de los recursos, en resumen lo que realizan es una descomposición balanceada de la señal. Estos mismos autores Rawski et al. en [80] y [81] comparan los resultados obtenidos en la FPGA de Altera Stratix III EP3SL50F484C2FPGA, con los que da la herramienta software Quartus 10.1 SP1 de la misma compañía y con una herramienta de la que dispone Altera específica para filtros con el software comercial Altera FIR Compiler 10.1.

En otro estudio, los autores Meyer–Baese et al. en [82] realizan una comparación de dos métodos para diseñar filtros: el método de aritmética distribuída comentado anteriormente de los autores Rawski et al. y un algoritmo sistemático introducido por Dempster y Macleod en [83] que le denominan *n–Dimensional Reduced Adder Graph* (RAG–n). En este trabajo se demuestra que RAG–n es un enfoque viable para la implementación de filtros cuando se utiliza RAG–n pipeline en bloques multiplicadores de las FPGA. Las FPGAs que utilizan son las de Altera con la herramienta *Altera FIR Compiler*. Comparando con el algoritmo de aritmética distribuída indican que se ha reducido sustancialmente el área utilizada. Lo ideal sería una combinación de estos dos algoritmos, ya que el algoritmo RAG–n obtiene mejores resultados en diseños con menor número de coeficientes, mientras que la aritmética distribuída ofrece mayores ventajas si existe un elevado número de coeficientes.

Otros autores que defienden el uso de la aritmética distribuída implementando la multiplicación en las LUT de que disponen las FPGAs son Sasao et al. en [84]. En este estudio definen una función de sumas ponderadas (WS) como un modelo matemático de la parte combinacional de la aritmética distribuída para filtros FIR en FPGAs de Altera. Esta función WS la realizan mediante una descomposición aritmética entre los bits más significativos y los menos significativos, que junto con el empleo de aritmética distribuída, optimiza los resultados tanto en área como en velocidad. Este método será eficiente cuanto mayor sea el número de bits de los operandos.

Por otro lado Mirzaei et al. en [85] proponen un algoritmo de desplazamiento y suma y lo comparan con la aritmética distribuida que utiliza el Coregen^{TM} de Xilinx, observando que existe una reducción del 50% en el número de *slices* utilizados, un 75% en el número de LUT y un 50% en el consumo de energía. Estos cálculos los realizan sobre la Virtex II y IV de Xilinx. Lo justifican indicando que los coeficientes del filtro son constantes, de

forma que siempre se realiza la multiplicación por una constante. Dividen el filtro en dos partes: un bloque multiplicador y un bloque de retardo, centrando su optimización en el bloque multiplicador. Las multiplicaciones por constantes las descomponen en sumas registradas y desplazamientos cableados.

Finalmente, para terminar los comentarios sobre el tema de los filtros comentaré el artículo de los autores Martínez et al. en [86]. Su propuesta principal consiste en un oscilador/contador autoconfigurable utilizando la misma frecuencia con que se están obteniendo los datos de entrada, con lo que los resultados del filtro los están obteniendo con latencia cero respecto al muestreo de la señal de entrada. Proponen una arquitectura con memorias circulares utilizando también los bloques multiplicadores de 18 bits de entrada y acumulador de 48 bits de que dispone la Virtex II y La Spartan III de Xilinx, además de los bloques DSP48 de la Virtex IV de Xilinx. Sus resultados los comparan también con los obtenidos por la herramienta software Xilin CoreGen™, concluyendo que se obtienen mejores resultados tanto en área como en frecuencia, especialmente para filtros con mayor número de coeficientes.

5.7. Otros estudios sobre la unidad MAC

Para terminar con el estudio de la bibliografía dedicada al tema de esta tesis, se describen brevemente dos artículos que se dedican solamente a la unidad MAC, sin estudiar explícitamente la convolución o sus aplicaciones (filtros).

Los autores Gierenz et al. en los artículos [87] y [88] describen la arquitectura y el hardware de una unidad MAC con parámetros para utilizarla en un *core* DSP empotrado escalable. Esta unidad MAC admite un amplio conjunto de instrucciones para tipos de datos enteros y fraccionarios. La generación de esta unidad es controlada por su arquitectura, por la aplicación y sus parámetros. En el proceso de generación de la unidad MAC sitúan los componentes adecuadamente asegurando una estimación del rendimiento rápido y fiable. Especialmente para las tecnologías actuales, donde los efectos del cableado influyen en el rendimiento de un circuito, un control estricto de la colocación de los componentes lleva a un predecible análisis cuantitativo y también a una posible optimización.

La arquitectura del core DSP que proponen es una arquitectura Harvard dual modificada escalable de punto fijo optimizada para aplicaciones empotradas. Los

operandos para las operaciones aritméticas son cargados y almacenados en un registro y los puertos de memoria de datos se utilizan para transferir datos entre la memoria y este registro. Utilizan un sistema *pipeline* con tres fases de la instrucción (búsqueda, decodificación y ejecución). El *core* de la arquitectura que proponen es escalable desde un amplia gama de parámetros como son: el número de palabras en el fichero de registros; tipos de datos soportados, su longitud de palabra y de su organización en el fichero de registro; tipo y número de unidades de ejecución; instrucciones soportados por unidad de ejecución; longitud de palabra y codificación de las instrucciones; y número de ciclos de reloj por fases de *pipeline*. Por otra parte la configuración del fichero de registros la diseñan para ser configurable como dos palabras o una doble palabra según requiera la aplicación, además de los bits de guarda. El acumulador forma parte del registro de datos en lugar de ser una registro dedicado como por ejemplo se emplea en Yuyun et al. [89]. La arquitectura de esta unidad MAC la dividen en cuatro etapas funcionales: generación de productos parciales, generación de la suma de productos, acumulación y la suma final. Hay que tener en cuenta que utilizan los datos en formato *carry–save*.

Por último comentaré un artículo que versa sobre la unidad MAC. Tan et al. en [90] proponen una arquitectura de un vector MAC en punto fijo de 64 bits capaz de soportar varias operaciones de multiplica–acumula: una de 64x64, dos de 32x32, cuatro de 16x16 u ocho de 8x8 bit con o sin signo; utilizando prácticamente el mismo hardware que una unidad escalar de 64 bits con solo un pequeño incremento en el área utilizada. Utilizan un método que ellos llaman “segmentación compartida” La arquitectura MAC escalar es vectorizada mediante la inserción de multiplexación en modo dependiente en el bloque de generación de los productos parciales y también cortando la cadena de acarreo en el árbol de sumadores y del sumador CPA final. Este vector MAC es eficiente en área y admite la segmentación (*pipeline*) lo cual lo hace adecuado para procesadores de alto rendimiento y procesadores dinámicamente reconfigurables.

Esta unidad MAC consiste en en generador de productos parciales que utiliza el algoritmo de Booth modificado, un sumador CSA en árbol de Wallace que consiste en compresores 3:2 (sumadores completos) y el sumador CPA final lo implementan mediante un *carry–look–ahead* (CLA) de 4 bits. Para reducir el vector de los productos parciales describen dos métodos: uno que consiste en la segmentación dividiendo los datos en vectores de 8 bits y otro método que consiste en introducir una señal (*kill*) que corta la

cadena de acarreo. Los resultados los implementan utilizando el lenguaje de descripción hardware Verilog en la herramienta software Synopsys Physical Compiler.

6 Optimización de arquitecturas para el cálculo de la convolución

RESUMEN

En este capítulo, se describen las arquitecturas propuestas para el cálculo de la convolución, centrándose en la arquitectura del MAC. Se establece, en primer lugar, una clasificación para operandos menores de 48 bits y para mayores de 48 bits, ya que es el tamaño de los multiplicadores o DSP empotrados que existen en las FPGAs actuales. Para operandos menores de 48 bits se propone una arquitectura del acumulador novedosa que consiste en un acumulador CSA–CPA combinado que optimiza el uso de los recursos en la FPGA Spartan–3 de Xilinx. Continúa el capítulo introduciendo unas arquitecturas específicas para los datos de entrada con el uso de buffers circulares para las dos formas de calcular la convolución, como son el algoritmo desde el punto de vista de la señal de salida y desde el punto de vista de la señal de entrada. Posteriormente se describirán las arquitecturas propuestas para el caso de operandos mayores de 48 bits. En primer lugar utilizando aritmética redundante y multiplicadores empotrados (tanto en la Spartan–3 como en la Spartan–6) y a continuación una implementación en la Spartan–6 utilizando compresores 5:3 y 6:3 con salida de doble acarreo. Termina el capítulo comparando los resultados y la descripción de las conclusiones.

6.1. Introducción

La operación de convolución es fundamental para muchas aplicaciones de procesamiento digital de la señal (filtrado, FFT, correlación, redes neuronales,...). Está basada en las operaciones de multiplicación y acumulación. Debido a su papel fundamental en aplicaciones DSP, muchos dispositivos FPGA de alto rendimiento han incorporado celdas especiales para procesamiento digital de la señal destinadas a realizar estas operaciones como son los multiplicadores empotrados y los bloques DSP descritos en el capítulo 4, para el caso de las FPGAs de Xilinx.

Para reducir el costo de las multiplicaciones implicadas en las aplicaciones con términos constantes, normalmente se implementa un multiplicador de constantes o se usa aritmética distribuida. No obstante, en algunas aplicaciones tales como filtros adaptativos, correlación y redes neuronales, la multiplicación es obligatorio realizarla. Esto ha obligado a los fabricantes a usar multiplicadores empotrados o bloques DSP para llevar a cabo este tipo de operaciones de forma eficiente.

La convolución implica muchas multiplicaciones y acumulaciones. Si una FPGA dispone de pocos multiplicadores, es frecuente hacer uso de ellos de forma iterativa para implementar esta función. Para aumentar el rendimiento, es necesario reducir el tiempo de ciclo especialmente cuando hay muchas operaciones. Dado que el tamaño del multiplicador es fijo en el dispositivo FPGA y el acumulador puede ser modificado por el usuario, un buen diseño del acumulador puede llevarnos a un mejor desarrollo de la operación en conjunto.

Muchos autores han implementado diferentes diseños para llevar a cabo la operación de convolución. Los actuales dispositivos FPGA incluyen lógica de acarreo rápida que permiten la implementación de sumadores con acarreo propagado rápido. Por tanto, en estos diseños utilizan sumadores de acarreo propagado, como por ejemplo en Wang et al. [49], Jamro & Wiatr [91], y Sriram & Kearney [92]. En las investigaciones realizadas en esta tesis se propone el uso de sumadores *carry–save* como una alternativa eficiente para reducir el tiempo de cálculo de la convolución. Está basado en una asignación a bajo nivel de los propios recursos del *slice* de la FPGA, que ofrece un rendimiento optimizado en comparación con la asignación automática (Junhyung et al.[93]).

Por otra parte, recientemente existe un creciente interés de la comunidad científica en el diseño de nuevos algoritmos que aprovechen la arquitectura interna de las FPGA tal y como se comentó en el artículo de Beuchat & Muller descrito en el capítulo anterior [11].

6.2. Arquitectura propuesta para la operación de convolución con operandos menores de 48 bits

En este apartado se presenta una arquitectura iterativa para el cálculo de la convolución, que es más rápida que anteriores implementaciones. Esto se consigue utilizando aritmética redundante, aunque suponga un leve incremento del hardware utilizado. Se desarrollan unas técnicas que permiten reutilizar los recursos hardware disponibles para obtener el resultado final en aritmética convencional.

La opción más eficiente para la realización de multiplicadores es utilizar los multiplicadores empotrados de los que disponen la mayoría de las FPGAs actuales. En el caso de que el ancho de los operandos de entrada no supere el ancho de palabra de los multiplicadores empotrados (que suele ser operandos de 18x18 y acumulador de 48 bits), el multiplicador se implementa en un solo multiplicador empotrado que genera salida convencional. A continuación se describe una arquitectura eficiente de este MAC para la Spartan–3 de Xilinx que dispone de LUT de 4 entradas y utiliza un único multiplicador empotrado (MULT18x18SIO) y aritmética redundante que publicamos en Moreno et al. [94].

En este estudio utilizamos solamente un multiplicador empotrado de la Spartan–3 de forma iterativa. La justificación de no utilizar más multiplicadores empotrados de este dispositivo puede encontrarse en Parandeh-Afshar et al. [68] y en Kuon & Rose [70]. En estos artículos se argumenta que no ofrecen mucha ventaja por dos razones: en primer lugar, porque se pierde tiempo en el ruteo de los datos entre estos bloques, y en segundo lugar por las pérdidas de recursos en las multiplicaciones (por ejemplo utilizar un multiplicador 18x18 para una multiplicación 9x9). Por tanto es preferible para pequeño tamaño de los datos utilizar la implementación basada en LUT por su flexibilidad, tal y como se describe a continuación.

La arquitectura que se propone para la convolución está basada en el uso iterativo de los multiplicadores empotrados incluidos en la mayoría de las FPGAs actuales. En la Fig. 6-1 se ilustra una arquitectura típica de la multiplicación y acumulación utilizando aritmética no redundante para la convolución de dos vectores, donde los operandos son de n bits y el acumulador tiene 2n + i bits (se añaden i bits para la extensión de signo y evitar el desbordamiento en las sumas). Se introduce un valor de cada vector en el multiplicador y su resultado se acumula con el anterior resultado parcial en cada iteración.

De esta manera, el tiempo de ciclo básico es:

$$\mathrm{Tcycle} = \mathrm{Tmult} + \mathrm{Tacc}$$

y el tiempo requerido para una operación es:

$$\mathrm{Ttotal} = \mathrm{N} * \mathrm{Tcycle}$$

siendo N el número de elementos del vector.

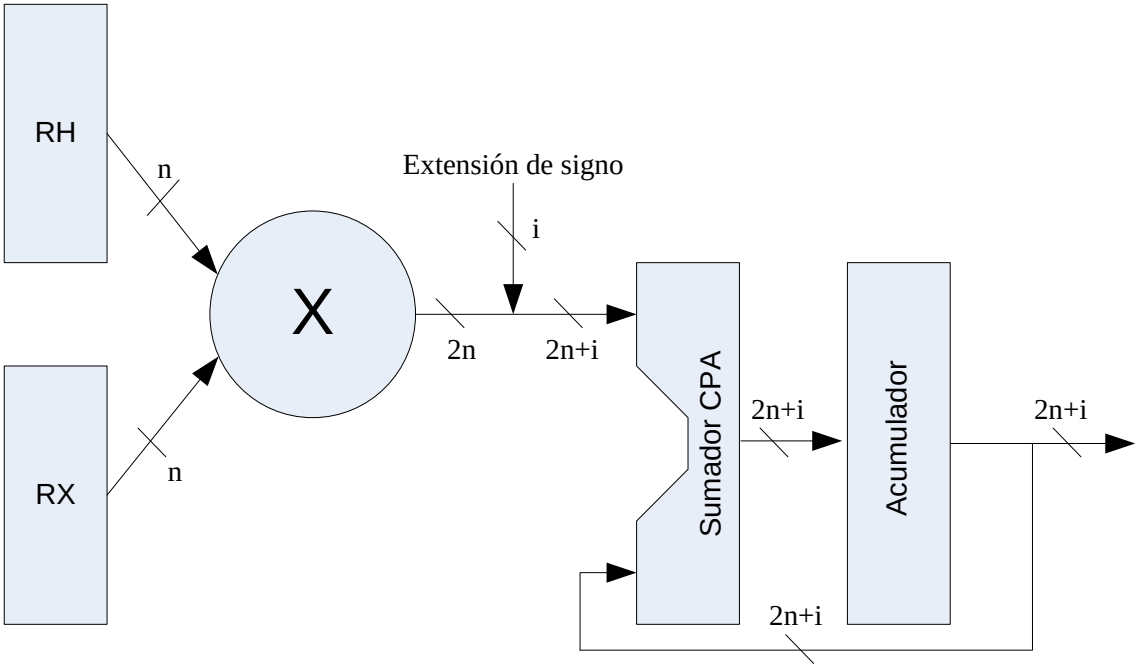


Figura 6-1 Arquitectura clásica para la operación de convolución.

Para reducir el tiempo de cálculo total (Ttotal) hay que reducir el tiempo de ciclo Tcycle. Los multiplicadores empotrados vienen predefinidos por el fabricante y se puede realizar *pipeline* para reducir su tiempo de ciclo efectivo, mientras que en el acumulador no se puede realizar *pipeline* debido a problemas de dependencia de datos. Para reducir el tiempo efectivo del acumulador se propone el uso de sumadores *carry–save* (CSA). Esta aritmética es especialmente útil cuando se tienen que realizar una gran cantidad de sumas, como es el caso del cálculo de la operación de convolución. No obstante, requiere una conversión final a la representación convencional que se suele realizar mediante un sumador con acarreo propagado (CPA). El retraso debido a CPA es mayor que el realizado con CSA, pero tiene una influencia pequeña respecto al cómputo del tiempo total si hay una gran cantidad de términos en la operación de convolución (que es lo más habitual). La principal desventaja es que un sumador CPA supone un incremento de hardware ya que se necesitan dos sumadores en total frente a un solo sumador que necesita la clásica implementación no redundante. La propuesta de esta tesis doctoral para solucionar este problema es el uso de un único sumador combinando un acumulador CSA + CPA.

Los resultados de este apartado fueron publicados por Moreno et al. en [94].

6.2.1. Arquitectura propuesta basada en un acumulador CSA–CPA combinado

La mayoría de las FPGAs actuales disponen de una circuitería especial dentro de sus *slices* para optimizar la implementación de sumadores con acarreo propagado. En las FPGA con dos LUTs por *slice*, cada *slice* es capaz de realizar la suma con acarreo propagado de dos bits. Si intentamos implementar sumadores *carry–save* en una FPGA, el bit de suma y el bit de acarreo deben calcularse en paralelo para todos los bits incluyendo los operandos de entrada, independientemente de los acarreos de entrada y salida. Desafortunadamente esto no es posible realizarlo en un único *slice* como se explica en *Beuchat and Muller* [11]. Esto significa que las herramientas de diseño hardware asignan dos LUTs (una para el cálculo del bit de suma, y otra para el bit de acarreo) cuando se implementa en lenguaje de descripción de hardware VHDL; esto es, se les asigna un *slice* completo para el cálculo de una etapa en lugar de dos que utiliza la suma en CPA.

Tradicionalmente el uso de la suma de CSA en FPGAs ha sido descartada debido a su ineficiente asignación en la estructura interna de estos dispositivos, como se indica en Jamro & Wiatr [91]. Las actuales FPGA incluyen lógica rápida de acarreo que permiten la implementación de sumadores con una rápida propagación del acarreo. Debido a ello, para longitudes de palabra pequeñas, los sumadores CPA son sólo algo más lentos que los CSA pero requieren muchas menos celdas lógicas en su implementación en FPGAs.

A pesar de ello, cuando la longitud de la palabra crece la diferencia de velocidad entre CSA y CPA también aumenta y convendría utilizar CSA a pesar del coste en aumento del área, hecho justificado por Ortiz et al. [36]. Sin embargo, cuando el número de sumas a realizar es significativamente mayor, el multiplicador se utiliza de forma repetitiva, con lo que el tiempo que se gana se multiplica por el número de sumas, y además el consumo en área es reducido. Por esta razón proponemos el uso de una arquitectura CSA para implementar la convolución de una palabra en serie.

La principal desventaja del uso de un sumador CSA es el consumo de área en la FPGA. Un sumador CSA utiliza el doble de registros que un CPA, y generalmente, necesita un sumador CPA al final para obtener el resultado en una representación convencional no redundante. A continuación se presenta un sumador CSA–CPA combinado que utiliza el área de un solo sumador.

En Fig. 6-2 se presenta la arquitectura completa para calcular la convolución basada en un acumulador CSA más un sumador CPA que calcula la suma final. Se trata de una

adaptación de la arquitectura clásica presentada en Fig. 6-1 para el caso de aritmética *carry–save*. Se puede observar que esta arquitectura iterativa utiliza los registros RC y RS para asignar las palabras de suma y acarreo en cada iteración, y se necesita un sumador final CPA para convertir el resultado a la representación convencional.

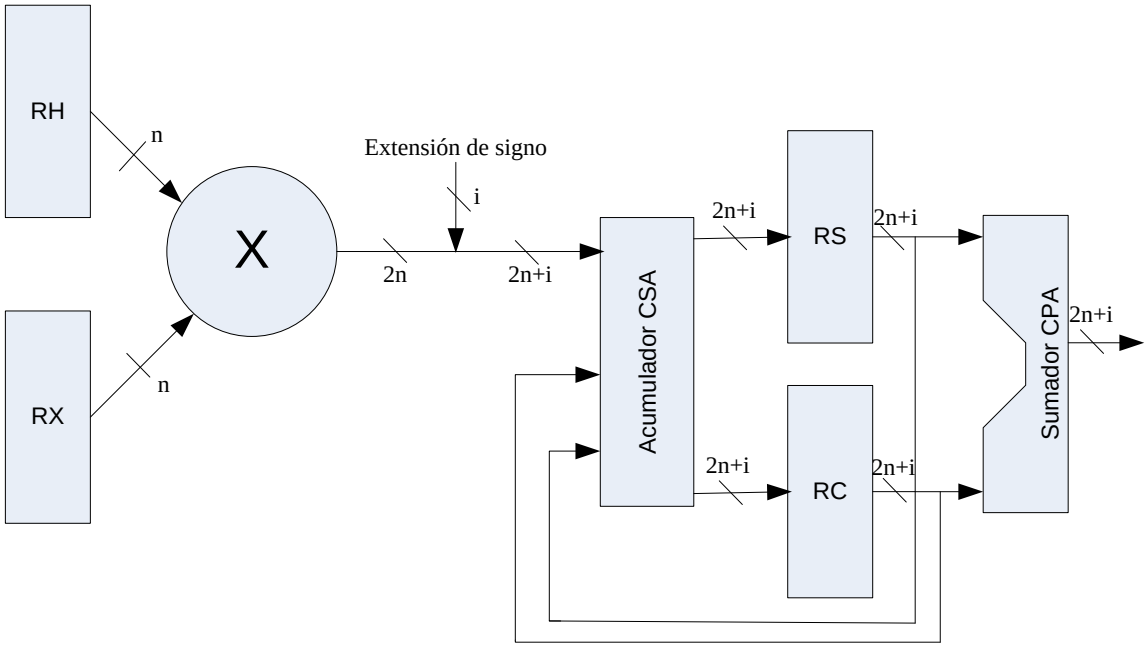


Figura 6-2 Convolución basada en CSA

La propuesta consiste en una arquitectura que elimina el área consumida del acumulador CPA final de la figura anterior. La mencionada arquitectura se muestra en Fig. 6-3.

El elemento clave es el acumulador CSA–CPA combinado que requiere el hardware de un solo acumulador CSA, sin añadir penalización temporal, que pasamos a describir en el siguiente apartado.

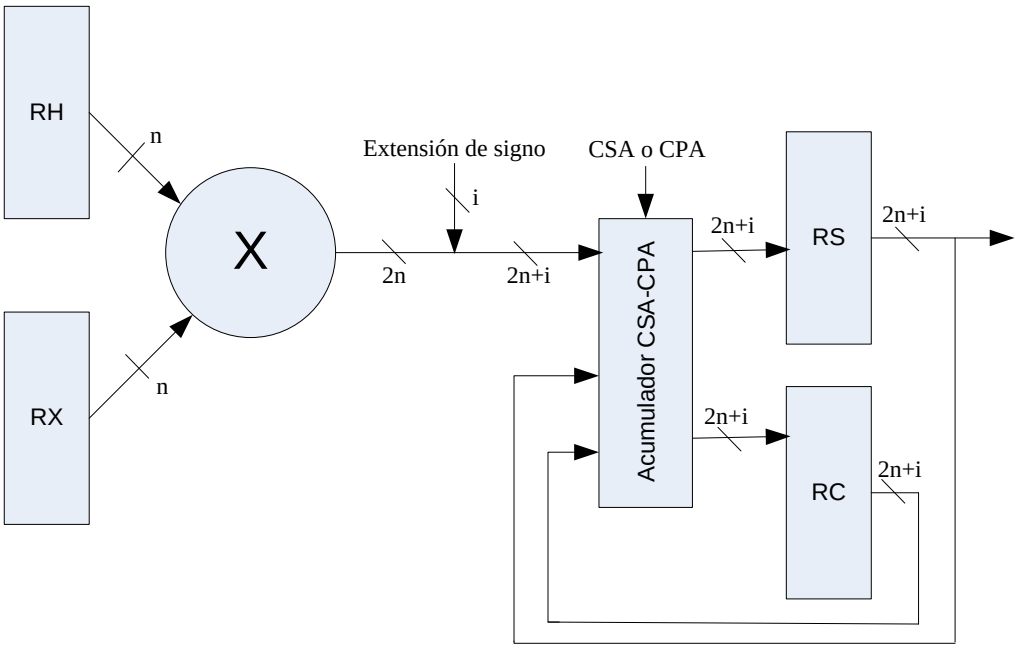


Figura 6-3 Convolución basada en CSA–CPA combinado

6.2.2. Acumulador CSA–CPA combinado en la Spartan–3

Las actuales FPGAs están principalmente diseñadas para aplicaciones del procesamiento digital de señales que impliquen operandos bastante pequeños (de 16 a 32 bits). Los fabricantes de FPGA decidieron incluir lógica de acarreo adicional para la implementación de rápidos sumadores *carry–ripple* para estos tamaños de operando. Supongamos por ejemplo la FPGA de bajo coste Spartan–3 de Xilinx. En la Fig. 6-4 se muestra la arquitectura simplificada de un slice de este dispositivo [41].

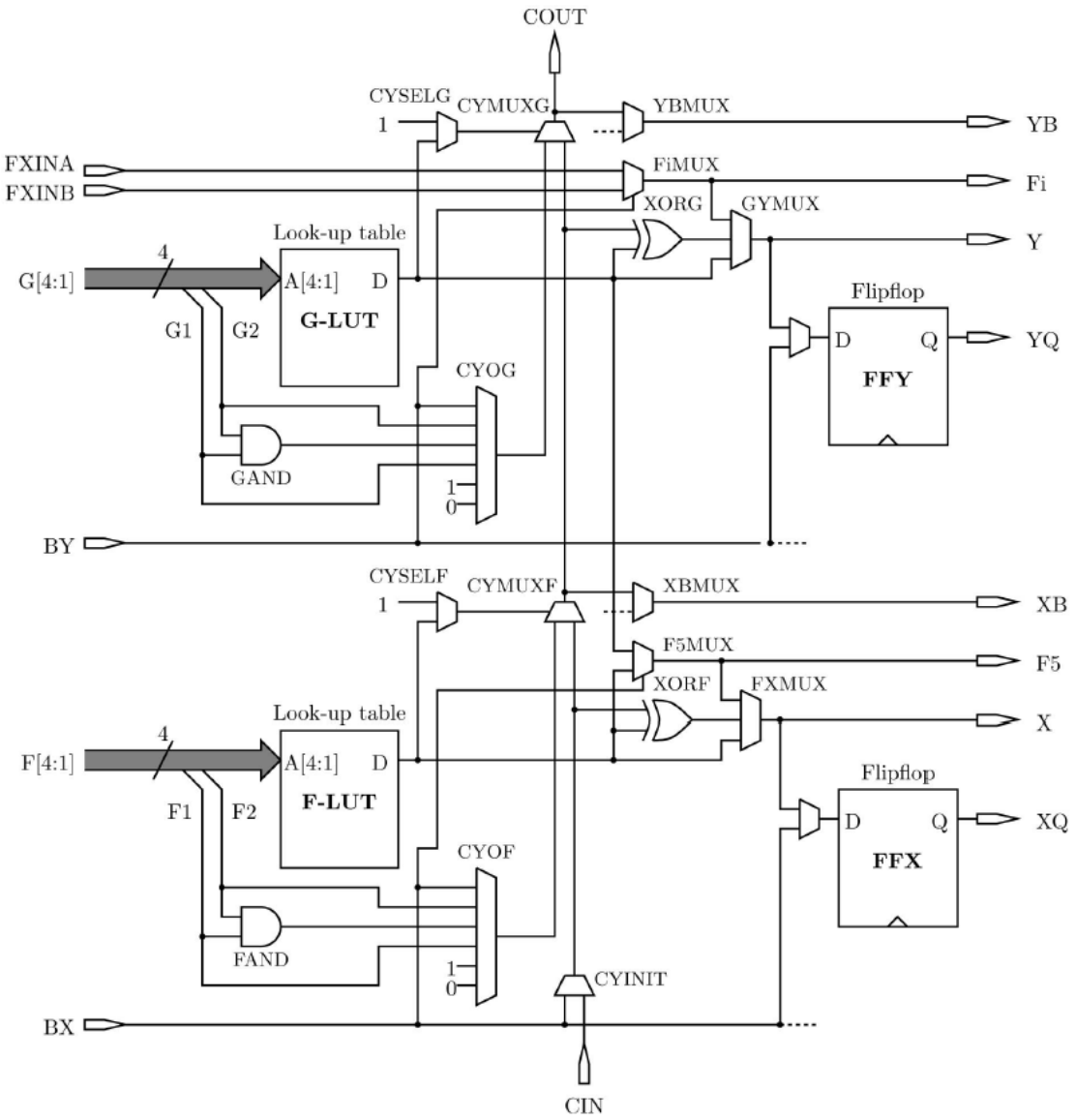


Figura 6-4 Diagrama simplificado de un *slice* de la FPGA Spartan–3 de Xilinx

Cada *slice* incluye dos generadores de función de cuatro entradas (*lookup table*, G–LUT and F–LUT), dos flip-flops (FFY and FFX), lógica de acarreo (CYSELG, CYMUXG, CYSELF, CYMUXF and CYINIT), puertas lógicas (GAND, FAND, XORG AND XORF), y multiplexores con varias funciones. Cada generador de función es implementado por medio de una tabla de búsqueda programable (*lookup table*, LUT).

Recordemos del capítulo 3, que un sumador completo (FA) es un circuito combinacional con tres bits de entrada (los bits a sumar x_i and y_i, y el acarreo de entrada c_{in}) y dos de salida (el bit de suma s_i y el acarreo de salida c_{out}). Tenemos que $s_i = x_i \oplus y_i \oplus c_{in}$, y el acarreo de salida $c_{out} = xi$, cuando $xi = yi$, y $cout = cin$ en cualquier otro caso. Supongamos que F–LUT calcula $xi \oplus yi$, entonces la puerta XORF obtiene el bit de suma s_i, mientras que el cálculo de el acarreo de salida c_{out} involucra a tres multiplexores

(CYOF, CYSELF y CYMUXF). El bit de suma s_i es enviado a otro *slice* (salida X) o almacenado en el flip–flop FFX. La G–LUT podría implementar un segundo sumador dentro del mismo *slice*, integrando así a dos bits de un sumador CRA. Algunos autores, como por ejemplo Jamro & Wiatr [91], afirman que en un sumador CSA convencional, es imposible implementar dos sumadores completos con acarreos de entrada independientes en el mismo *slice*, ya que cada *slice* dispone solamente de una sola entrada de acarreo, con lo que se requeriría el doble de recursos hardware. Por lo tanto, las herramientas de diseño hardware asignan dos *slices* cuando se hace una descripción VHDL de s_i y c_{out} dejando sin utilizar a G–LUT.

En la Fig. 6-5 se describe la arquitectura simplificada de un *slice* implementando un sumador con acarreo propagado (CPA).

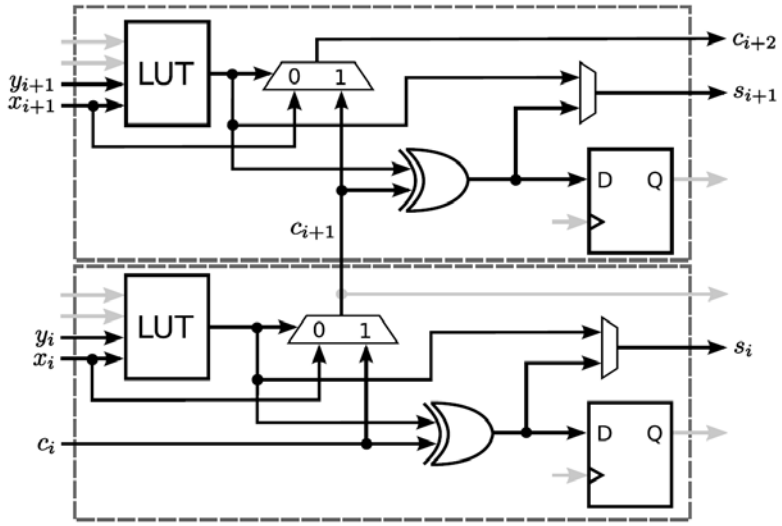


Figura 6-5 Diagrama simplificado de un *slice* implementando un CPA

A continuación se describe la estructura del acumulador CSA–CPA. En las Fig. 6-6 y 6-7 se muestra la funcionalidad del acumulador combinado CSA–CPA (bloque CSA–CPA de la Fig. 6-3), donde las líneas grises representan las señales que no intervienen en la operación. En la Fig. 6-6 se muestra el acumulador trabajando en modo CSA, mientras que la Fig. 6-7 describe el modo CPA. Xi representa un bit de la salida del multiplicador mientras que Ci y Si representan los bits de suma y acarreo respectivamente. La entrada Sel selecciona el modo CSA (Sel=0) o el modo CPA (Sel=1).

En modo CSA la tabla de búsqueda F–LUT es la encargada de realizar la suma de Xi y Si, y la puerta XOR inferior calcula el bit de suma final (almacenado en el flip–flop

correspondiente). El bit de acarreo es transmitido a través del multiplexor inferior y almacenado en el registro superior ya que Sel=0 pone a cero la salida de la tabla de búsqueda G–LUT. Csi también se transmite utilizando el multiplexor superior. En modo CPA el bit de suma es generado a partir de C_{Si-1} , Si y Cpi−1 por medio de F–LUT y la puerta XOR asociada, mientras que el acarreo es transmitido al siguiente *slice* cruzando ambos multiplexores (la salida de G–LUT se pone a uno si Sel=1).

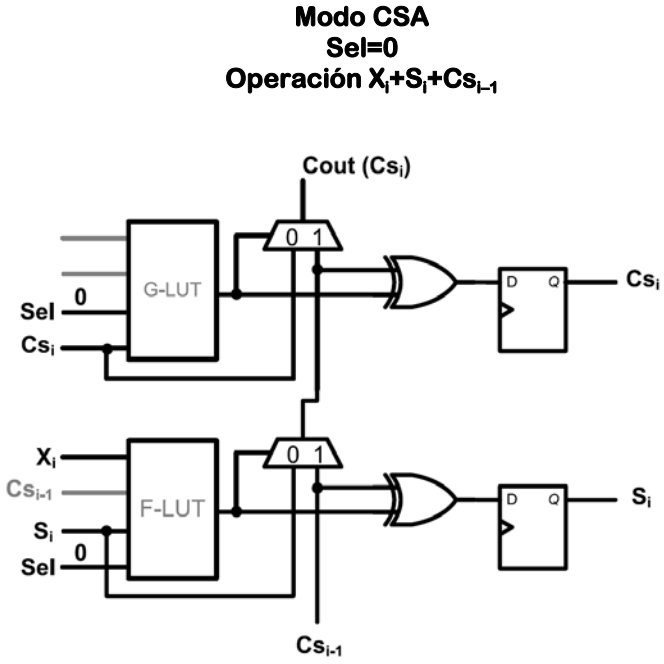


Figura 6-6 Acumulador combinado CSA–CPA en modo CSA

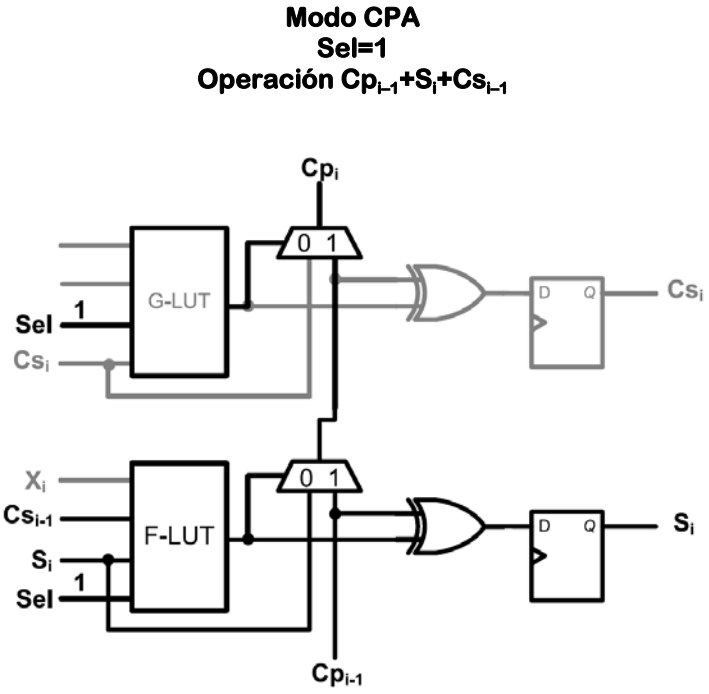


Figura 6-7 Acumulador combinado CSA–CPA en modo CPA

6.2.3. Resultados en la Spartan–3

En esta sección se describe una comparación cuantitativa entre las arquitecturas presentadas anteriormente. Estos diseños han sido simulados, evaluados y sintetizados usando Mentor Graphics ModelSim, Xilinx ISE 9.1i y la FPGA de Xilinx Spartan XC3S500e-5. Aunque actualmente han aparecido nuevos modelos de FPGA, las estudiadas en esta tesis basadas en *look up table* son las más utilizadas para aplicaciones de bajo coste. Las implementaciones propuestas usan los multiplicadores empotrados que poseen las actuales FPGAs. De esta manera, optimizamos los tiempos y los recursos del sistema.

Se han estudiado dos casos: la convolución para entradas de 18 bits (que usan los multiplicadores empotrados de 18x18 bits) con acumuladores de 48 bits, y para entradas de 35 bits (multiplicadores de 35x35) con un acumulador de 84 bits.

6.2.3.1. Convolución con operandos de 18 bits y acumulador de 48 bits

Los resultados de la implementación para la convolución con datos de entrada de 18 bits se muestran en la tabla 6-1.

Tabla 6-1. Resultados de la implementación para multiplicadores de 18x18

Arquitectura	Fq (MHz)	LUTs	SLICEs
C–CPA	114	48	24
C–CSA	149	144	72
C–CSA–CPA	149	96	48

Estas arquitecturas usan multiplicadores empotrados de 18x18 y un acumulador de 48 bits implementado usando lógica estándar. Se han evaluado tres posibilidades diferentes: C–CPA, donde se utiliza para la acumulación un sumador con acarreo propagado estándar (ver Fig. 6-1); C–CSA, donde se utiliza un sumador *carry–save* para las sumas parciales y un sumador con acarreo propagado para realizar la suma final y obtener así el resultado en representación estándar (ver Fig. 6-2); y C–CSA–CPA, donde se utiliza el acumulador combinado CSA–CPA explicado en el apartado anterior. Se muestra en la tabla 6-1 el número de *slices* y *look up table* utilizados, y la frecuencia máxima alcanzada para cada implementación de la convolución. Es importante señalar que en los diseños C-CSA and C-CSA-CPA se incluye el área total utilizada pero no los retardos, porque sólo afecta a la última iteración y, si el número de operandos es lo suficientemente grande (que es el caso que sucede en la práctica), su influencia en el retardo total es despreciable.

Como era de esperar, con el diseño CPA se obtiene el mejor área, pero también el peor retardo. Es importante señalar que el retardo correspondiente al acumulador es casi el doble del retardo del multiplicador. Esto se debe a la utilización de los multiplicadores internos de la FPGA. Gracias a la utilización de la representación *carry–save*, el retardo del acumulador en las otras dos arquitecturas es similar a la del multiplicador, lo cual permite una importante mejora de velocidad: cerca de un 30% más rápido.

No obstante, este aumento de la frecuencia se logra a costa de un importante aumento de área. Señalar que el número de slices de la arquitectura C–CSA es tres veces mayor que en la arquitectura C–CPA, mientras que el área de la arquitectura C–CSA–CPA es dos veces mayor que su correspondiente C–CPA, y 33% menos que la de C–CSA.

Como conclusión, la arquitectura propuesta C–CSA–CPA tiene el mismo retardo que la arquitectura basada en un puro CSA y utiliza un 33% menos de área. Ambas arquitecturas basadas en CSA tienen una velocidad de alrededor de un 30% mejor en comparación con la clásica CPA.

6.2.3.2. Convolución con operandos de 36 bits y acumulador de 84 bits

Seguidamente se ha realizado la implementación con datos de entrada de 36 bits (multiplicadores de 36x36 bits) con un acumulador de 84 bits. En este caso los multiplicadores empotrados no pueden ser utilizados directamente y hemos utilizado la herramienta Coregen de Xilinx para generar los multiplicadores de 36x36 bits (Coregen utiliza cuatro multiplicadores de 18x18 bits). Dado que el objetivo es acelerar el cálculo, el multiplicador tiene siete etapas *pipeline* (es el valor generado por la herramienta software; de hecho, la velocidad no aumenta si se utilizan más de siete estados). En consecuencia, si la convolución requiere N multiplicaciones (N operandos cada vector), la latencia es N+7 ciclos y el rendimiento corresponde a un resultado cada N ciclos.

En la tabla 6-2 se muestran los resultados de la implementación de los diferentes diseños para operandos de entrada de 36 bits.

Tabla 6-2. Resultados de la implementación para multiplicadores de 36x36 con 7 etapas *pipeline*

Arquitectura	Fq (MHz)	LUTs	SLICEs
C–CPA	132,4	498	531
C–CSA	166,5	665	615
C–CSA–CPA	166,5	581	573

Como en el caso de la entrada de 18 bits, se obtiene una importante ventaja en la velocidad de la convolución (alrededor del 27%) similar al caso de 18 bits. En principio, se espera una mejor velocidad ya que el retardo del CSA no aumentará con un operando de mayor anchura. La razón de este comportamiento es la siguiente: con operandos de 36 bits la velocidad del sistema viene impuesta por el multiplicador empotrado.

No obstante, el incremento de área ha bajado considerablemente respecto al caso de datos de entrada de 18 bits, ya que los siete estados del multiplicador de 36x36 requieren demasiado hardware para todos los diseños. De esta manera, la influencia del CSA y CPA en el área de la arquitectura total es relativamente menor en el caso de estos diseños. Para ser exactos, el incremento en área para la arquitectura C–CSA es cerca del 15% y para la arquitectura C–CSA–CPA es sólo del 8% (se obtiene una reducción del 7% al comparar ambas arquitecturas).

En conclusión, para operandos de entrada de 36 bits la arquitectura basada en un CSA–CPA combinado tiene una velocidad similar al de 18 bits (27%) con un coste en hardware pequeño (8%) en comparación con la implementación basada solo en CPA. Resultados similares en tiempo se obtienen para las arquitecturas basadas en CSA más CPA con un ligero incremento de área.

6.2.3.3. Conclusiones de estos resultados

En este apartado se han presentado varias arquitecturas para realizar el cálculo de la convolución, utilizando CSA para acelerar dicho cálculo, y especialmente adaptado a las arquitecturas internas de los dispositivos FPGA. Los diseños se han adaptado a la estructura de la FPGA lo cual ha permitido aprovechar al máximo los recursos del dispositivo. Las arquitecturas se han comparado entre ellas desde un punto de vista cualitativo y cuantitativo y se han mostrado los resultados obtenidos sobre una FPGA concreta.

Se ha propuesto una arquitectura basada en un acumulador combinado CSA–CPA reutilizando el hardware requerido por el CSA de tal manera que ninguna necesita la implementación hardware del CPA necesario para la conversión final de aritmética redundante a aritmética convencional. Esto conlleva al mimo tiempo de cálculo que el obtenido utilizando un CSA más un CPA final. En otras palabras, la conversión final de

aritmética redundante a aritmética convencional no supone un coste hardware para la arquitectura propuesta CSA–CPA combinada (ver Figs. 6-2 y 6-3).

De los resultados experimentales se concluye que para las dos arquitecturas presentadas basadas en CSA se obtienen mejores resultados temporales (aproximadamente el 30%), y el acumulador combinado CSA–CPA consigue una importante reducción en hardware comparada con la arquitectura CSA más un CPA final. Cuando se utilizan datos de entrada mayores y se necesita mayor velocidad es necesario utilizar una estructura *pipeline* de los multiplicadores. En este caso, la velocidad y el hardware adicional exigido por el CSA tiene una relativa menor importancia en el área del sistema total debido al gran hardware requerido por los multiplicadores (solo un 8% más hardware que un diseño CPA con casi un 30% más de velocidad para datos de entrada de 36 bits).

Existe un amplio espectro de aplicaciones en las cuales la idea de un acumulador CSA–CPA combinado puede ser utilizado, muchos algoritmos del procesamiento digital de señales están basados en la multiplicación y acumulación. Consecuentemente, una cantidad enorme de las actuales arquitecturas basadas en FPGA pueden ser rediseñadas utilizando la técnica propuesta para aumentar su rendimiento.

6.3. Arquitectura con buffers circulares para los algoritmos de convolución

La arquitectura propuesta en el apartado anterior presenta una arquitectura iterativa para el cálculo de la convolución más rápida que otras implementaciones. Esto se consigue mediante el uso de aritmética redundante que, aunque implica un leve incremento del hardware, se ha desarrollado una técnica que permite reutilizar los recursos hardware para obtener el resultado final en aritmética convencional. La arquitectura propuesta para el cálculo de la convolución se ha basado en la utilización reiterativa de los multiplicadores empotrados de las FPGAs de bajo coste actuales (Spartan–3 de Xilinx). En ese apartado se había propuesto un acumulador combinado CSA–CPA utilizando la lógica de acarreo cuando se implementan sumadores *carry–save*. Este acumulador combinado CSA–CPA requiere el hardware de un acumulador CSA sólamente, sin añadir penalización temporal. Pero no se describió la forma en que se trataban los registros donde se almacenaban las señales que se iban a convolucionar.

En este apartado se propone una nueva arquitectura basada en la anterior, que incluye el uso de *buffers* circulares para registrar las muestras de las señales de entrada o de salida

dependiendo del algoritmo que vayamos a utilizar para el cálculo: desde el punto de vista de la entrada (*input side algorithm*) o desde el punto de vista de la salida (*output side algorithm*), que fueron descritos en el capítulo 2 de esta tesis.

En el procesamiento digital de señales, la convolución puede ser entendida de dos formas distintas: desde el punto de vista de la señal de entrada y desde el punto de vista de la señal de salida [5]. Desde el punto de vista de la señal de entrada implica analizar cómo cada muestra de la señal de entrada contribuye en muchos valores de la señal de salida, proporciona una comprensión conceptual de cómo la convolución influye en el procesamiento digital de señales. Desde el punto de vista de la señal de salida evalúa cómo cada muestra en la señal de salida recibe información de muchas muestras de la señal de entrada.

Supongamos por ejemplo la convolución de dos señales x[n] y h[n] de longitudes M = 5 y N = 4.

$x[n] = [x_4, x_3, x_2, x_1, x_0]$ de longitud M = 5

$h[n] = [h_3, h_2, h_1, h_0]$ de longitud N =4

Por definición de convolución discreta:

$$y[i] = x[n] * h[n] = \sum_{j=0}^{M+N-1} x[j] \cdot h[i-j]$$

de longitud M + N – 1 = 5 + 4 – 1 = 8

Cada una de las muestras de la señal de salida, $y[n] = \{y_7, y_6, y_5, y_4, y_3, y_2, y_1, y_0\}$, se calcularía de la forma siguiente:

$y_0 = x_0 \cdot h_0$

$y_1 = x_0 \cdot h_1 + x_1 \cdot h_0$

$y_2 = x_0 \cdot h_2 + x_1 \cdot h_1 + x_2 \cdot h_0$

$y_3 = x_0 \cdot h_3 + x_1 \cdot h_2 + x_2 \cdot h_1 + x_3 \cdot h_0$

$y_4 = x_1 \cdot h_3 + x_2 \cdot h_2 + x_3 \cdot h_1 + x_4 \cdot h_0$

$$y_5 = x_2 \cdot h_3 + x_3 \cdot h_2 + x_4 \cdot h_1$$

$$y_6 = x_3 \cdot h_3 + x_4 \cdot h_2$$

$$y_7 = x_4 \cdot h_3$$

6.3.1. Algoritmo desde el punto de vista de la salida

El primer punto de vista (*the output side algorithm*) realiza el cálculo de la convolución de las ecuaciones anteriores en horizontal (por filas). Examina como cada muestra de la señal de salida recibe información de muchos puntos de la señal de entrada. Este punto de vista describe la matemática de la operación de convolución. Puede calcularse el valor de cada muestra de la señal de salida independientemente del valor de las otras muestras.

El algoritmo utilizado para este cálculo se indica a continuación:

```
For  i=0  to  M+N-1
    y[i] = 0
    For  j=0  to  N
    If (i-j ≥ 0)
    If (i-j < M)
          y[i] =  y[i]  +  x[j] · h[i-j]
    End for
End for
```

La arquitectura propuesta para este cálculo se describe en la figura 6-8.

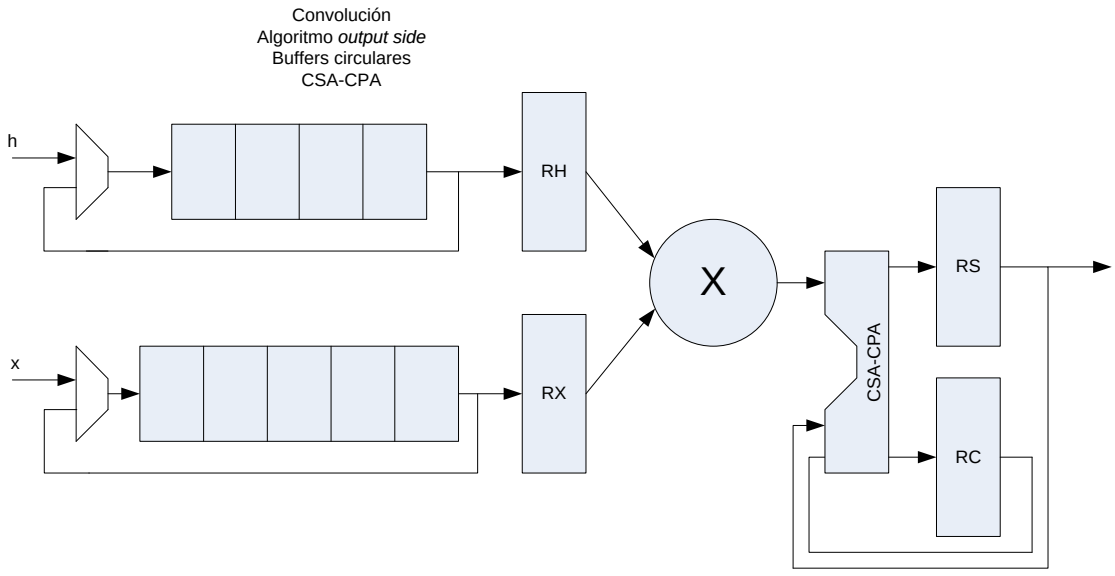


Figura 6-8 Arquitectura propuesta para el algoritmo *output side*

En esta arquitectura se han introducido dos *buffers* circulares para las muestras de cada señal de entrada x[n] y h[n] que se almacenan en los registros RX y RH, un multiplicador realiza su producto y para realizar la suma utilizamos el acumulador CSA–CPA combinado descrito en el apartado anterior (Figuras 6-6 y 6-7).

6.3.2. Algoritmo desde el punto de vista de la entrada

El segundo punto de vista (*the input side algorithm*) realiza el cálculo de la convolución de las ecuaciones anteriores en vertical (por columnas). Analiza como cada muestra en la señal de entrada afecta a varias muestras de la señal de salida, es decir, toma x_0 y la multiplica por todas las muestras de la respuesta al impulso h_3, h_2, h_1, h_0. En el primer ciclo calcularía $y_0 = x_0 \cdot h_0$, a continuación va calculando los términos producto de cada muestra de salida: $x_0 \cdot h_1$, $x_0 \cdot h_2$, $x_0 \cdot h_3$, etc. Posteriormente toma x_1 y realiza el producto por todas las entradas h; es decir, calcula $x_1 \cdot h_0$, que al sumarlo con su primer término producto ya calculado obtiene la segunda muestra de la señal de salida y_1, calcula seguidamente $x_1 \cdot h_1$, $x_1 \cdot h_2$, $x_1 \cdot h_3$, etc. Continúa las operaciones con todas las muestras de la señal de entrada.

Este punto de vista de la convolución está basado en el concepto fundamental del procesamiento digital de señales: descompone la entrada, pasa los componentes por el sistema y sintetiza la salida.

El algoritmo que describe este cálculo es el siguiente:

```
For  i=0  to  M+N–1
    For  j=0  to  M
         y[i+j] =  y[i+j]  +  x[i] · h[j]
    End for
End for
```

y la arquitectura propuesta para este caso es la siguiente:

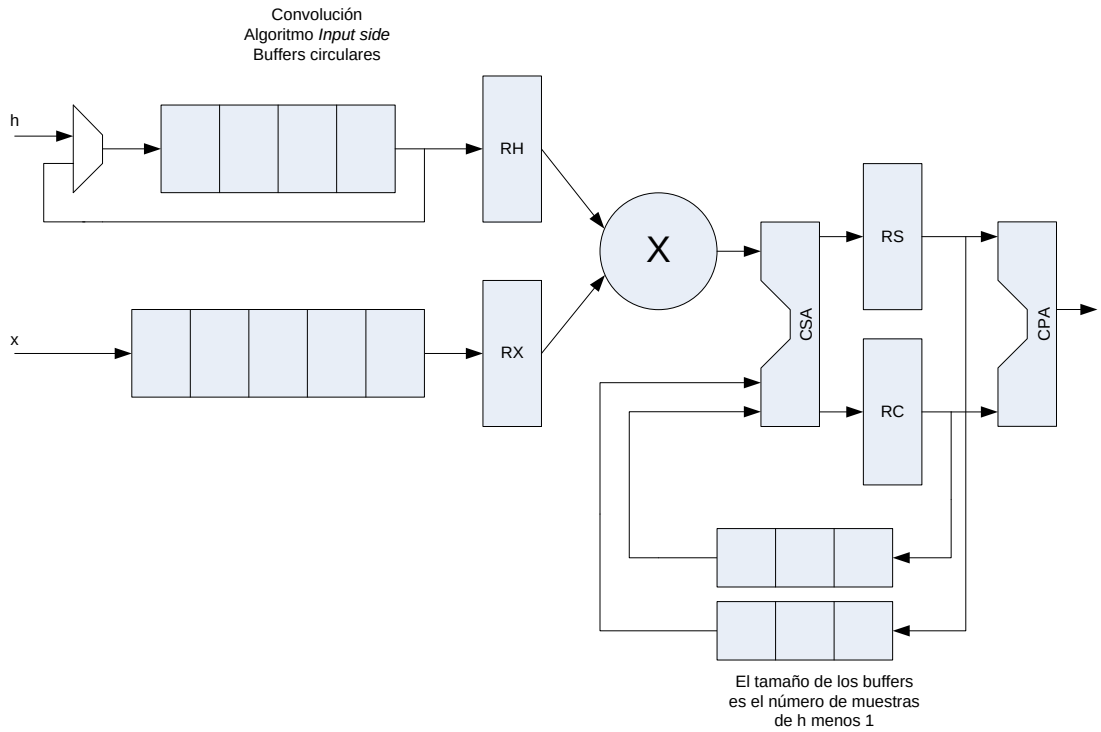


Figura 6-9 Arquitectura propuesta para el algoritmo *input side*

En este caso se ha sustituído el acumulador CSA–CPA combinado por un acumulador CSA y dos *buffers* para las salidas de los registros de la suma y de los acarreos (RS y RC respectivamente), pero para obtener la suma final hay que añadir un sumador CPA convencional que realice la suma final como determina la aritmética *carry–save*. Podemos observar que esta estructura iterativa utliza los registros RC y RS para almacenar las palabras de suma y acarreo en cada iteración, y que es necesario un sumador CPA final para convertir el resultado de aritmética redundante a representación convencional. La longitud de los buffers de RS y RC debe ser el número de muestras de h[n] menos uno.

6.3.3. Resultados experimentales de los buffers circulares

Estos diseños han sido simulados, evaluados y sintetizados utilizando Mentor Graphics ModelSim, Xilinx ISE 9.1i y la FPGA de Xilinx Spartan–3 XC3S500E–5. A pesar de que otros modelos de FPGA han surgido recientemente, los modelos basados en tablas de búsqueda (LUT) estudiados en este apartado son actualmente los más utilizados para aplicaciones de bajo coste. Las implementaciones propuestas utilizan los multiplicadores empotrados de las FPGAs actuales, y se han optimizado los tiempos y los recursos del dispositivo.

Los resultados de la implementación de la convolución de datos de entrada de 18 bits se muestran en la tabla 6-3 . En estas arquitecturas se ha utilizado un multiplicador empotrado de 18x18 y un acumulador de 48 bits implementado mediante lógica estándar.

Tabla 6-3. Resultados de la implementación para multiplicadores de 18x18 y 16 muestras

Arquitectura	Fq (MHz)	LUTs
C–CPA	113,4	84
Input side	148,5	168
Output side	148,1	132

Por otro lado, una LUT de un *slice* de la FPGA Spartan–3 se puede configurar como un registro de desplazamiento de 16 bits sin utilizar el flip–flop disponible de cada slice. Por tanto, cada *buffer* de entrada o de salida utilizará una *look–up–table* si consta de 16 muestras o menos, dos LUTs si las muestras están entre 16 y 32, y así sucesivamente. El número de LUT mostrado en la tabla 6-3 corresponde al caso de un número de muestras menor o igual que 16. En el caso de que h[n] o x[n] sea mayor que 16 hay que añadir a los resultados 18x2 LUT por cada número de muestras que se superen múltiplo de 16. Además en el caso de la implementación *input side* hay que sumar otras 18x2 LUT extras por cada número de muestras de h[n] que se superen múltiplo de 16, ya que el tamaño de los *buffers* de salida son el número de muestras de h menos uno.

En este apartado se ha propuesto una arquitectura CSA–CPA combinada para el algoritmo desde el punto de vista de la salida basándonos en la reutilización del hardware requerido por el CSA de forma que no se añade hardware adicional para la suma final CPA necesario para la conversión de aritmética redundante a aritmética convencional. Se emplea el mismo tiempo de cálculo que con el uso de CSA más un CPA final para el algoritmo *input side*. En otras palabras, la conversión final de aritmética redundante a convencional no tiene coste hardware para la arquitectura CSA–CPA propuesta. Por otro lado, el uso de buffers circulares no añade retraso significativo para el cálculo final, aunque se consigue utilizando los recursos principales de la FPGA.

Los resultados de este apartado han sido publicados en Moreno et al. [47]

6.4. Arquitectura del MAC para anchos de palabra mayores de 48 bits utilizando aritmética redundante y multiplicadores empotrados

En este apartado y el siguiente, se van a describir las arquitecturas propuestas para la implementación de la unidad multiplica–acumula (MAC) cuando los valores de entrada

son mayores de 18x18 bits y el acumulador excede los 48 bits, por ser los valores que tienen los multiplicadores de las FPGAs de bajo coste de Xilinx que se emplean en los resultados de esta tesis doctoral: MULT18x18SIO para la Spartan–3 y DSP48A1 de la Spartan–6, descritos en el capítulo 4.

La arquitectura genérica de una unidad MAC debe de constar de un multiplicador, de un sumador y de un acumulador, tal y como se describe en la figura 6-10:

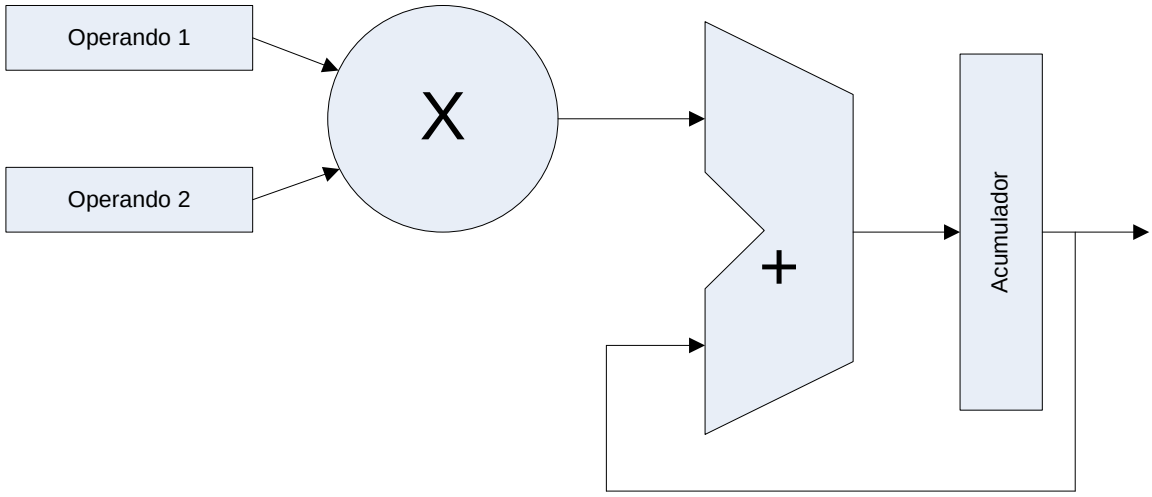


Figura 6-10 Arquitectura genérica de un MAC

En los apartados anteriores se ha estudiado que para la realización de multiplicadores es más conveniente la utilización de los multiplicadores empotrados incluídos en la mayoría de las FPGAs actuales. Cuando el ancho de los operandos es menor de 48 bits el multiplicador se implementa utilizando un solo multiplicador empotrado que genera salida convencional. En los apartados anteriores se describió una arquitectura eficiente para la Spartan–3 de Xilinx que dispone de LUT de 4 entradas.

En el caso de la Spartan–6, se dipone de *look–up table* de 6 entradas (LUT 6), y además integran de fábrica bloques DSP, el DSP48A1, que contienen entre otros recursos, un MAC de 48 bits, y no contienen multiplicadores aislados como era el caso de las LUT 4 de la Spartan–3. En la Spartan–6 la forma más eficiente para diseñar un MAC de 48 bits o inferior es utilizar estos bloques DSP.

En el caso de que el ancho de palabra de los operandos supere el ancho de los multiplicadores empotrados o bloques DSP, una opción eficiente con salida CSA se presenta en un *report* interno de nuestro grupo de investigación que se describe en Ortiz et

al. [95]. En este estudio se describe el diseño de multiplicadores *carry–save* en FPGAs para distintos anchos de palabra. La arquitectura genérica de una unidad MAC utilizando multiplicadores CSA se muestra en la figura 6-11, en la que se ha modificado la de la figura 6-10.

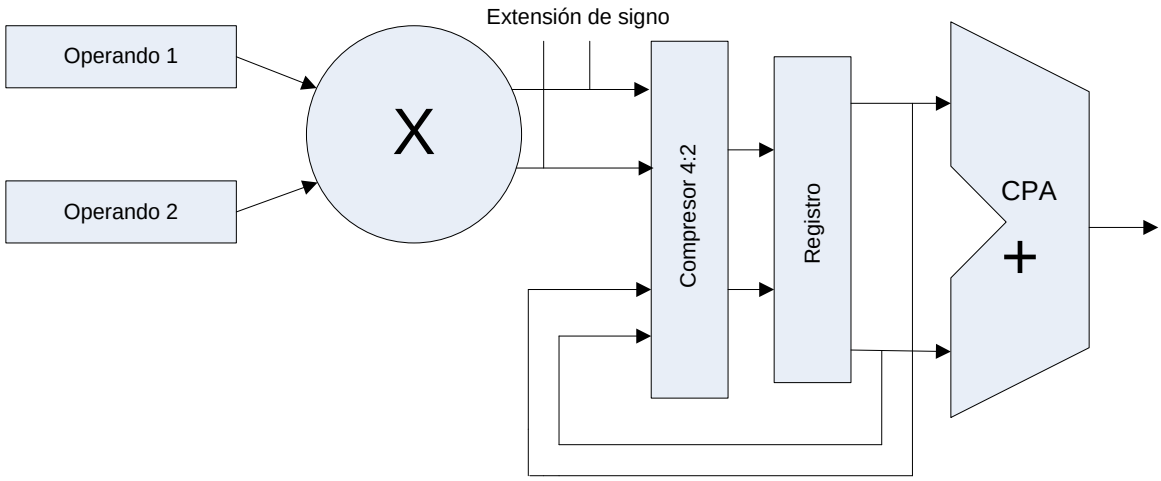


Figura 6-11 Arquitectura genérica de un MAC en CSA

El diagrama lógico de un compresor 4:2 se muestra en la figura 6-12.

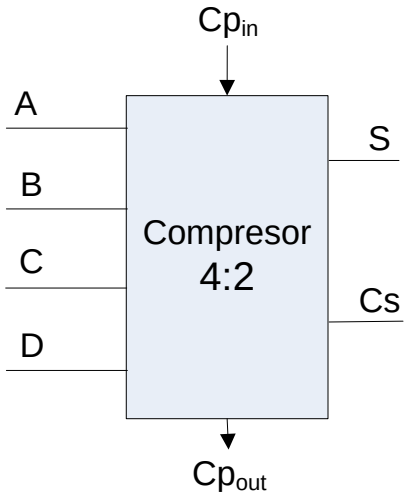


Figura 6-12 Símbolo lógico de un compresor 4:2

El compresor 4:2 puede generarse con primitivas a partir de su tabla de verdad como se indica en la tabla 6-4, o bien a partir de dos sumadores completos (o compresores 3:2) como se indica en la figura 6-13. A la hora de describir el comportamiento mediante la tabla de verdad, se debe buscar que el acarreo propagado finalice. Por ejemplo en la fila correspondiente a las entradas 10001 habría dos opciones 010 ó 100 se opta por la primera para que finalice el acarreo y pasa a la salida de acarreo Cs, de esta forma el acarreo

propagado de salida no depende del acarreo propagado de entrada. En el caso de la fila 11111 el acarreo de entrada se lleva al acarreo almacenado Cs y el propagado se genera a partir de las entradas, de esta forma se consigue cortar el acarreo. El acarreo de entrada se lleva al acarreo Cs.

Tabla 6-4. Tabla de verdad de un compresor 4:2

Cp_{in}	D	C	B	A	Cp_{out}	Cs	S
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	0	0	1	0
0	0	1	1	1	0	1	1
0	1	0	0	0	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	0	1	1	0	1	1
0	1	1	0	0	0	1	0
0	1	1	0	1	0	1	1
0	1	1	1	0	0	1	1
0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1
1	0	0	0	1	0	1	0
1	0	0	1	0	0	1	0
1	0	0	1	1	0	1	1
1	0	1	0	0	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	0	0	1	1
1	0	1	1	1	1	1	0
1	1	0	0	0	0	1	0
1	1	0	0	1	0	1	1
1	1	0	1	0	0	1	1
1	1	0	1	1	1	1	0
1	1	1	0	0	0	1	1
1	1	1	0	1	1	1	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1

En la figura 6-13 se muestra como se construye un compresor 4:2 a partir de compresores 3:2 (o sumadores completos) como se explica en Ercegovac & Lang en [10].

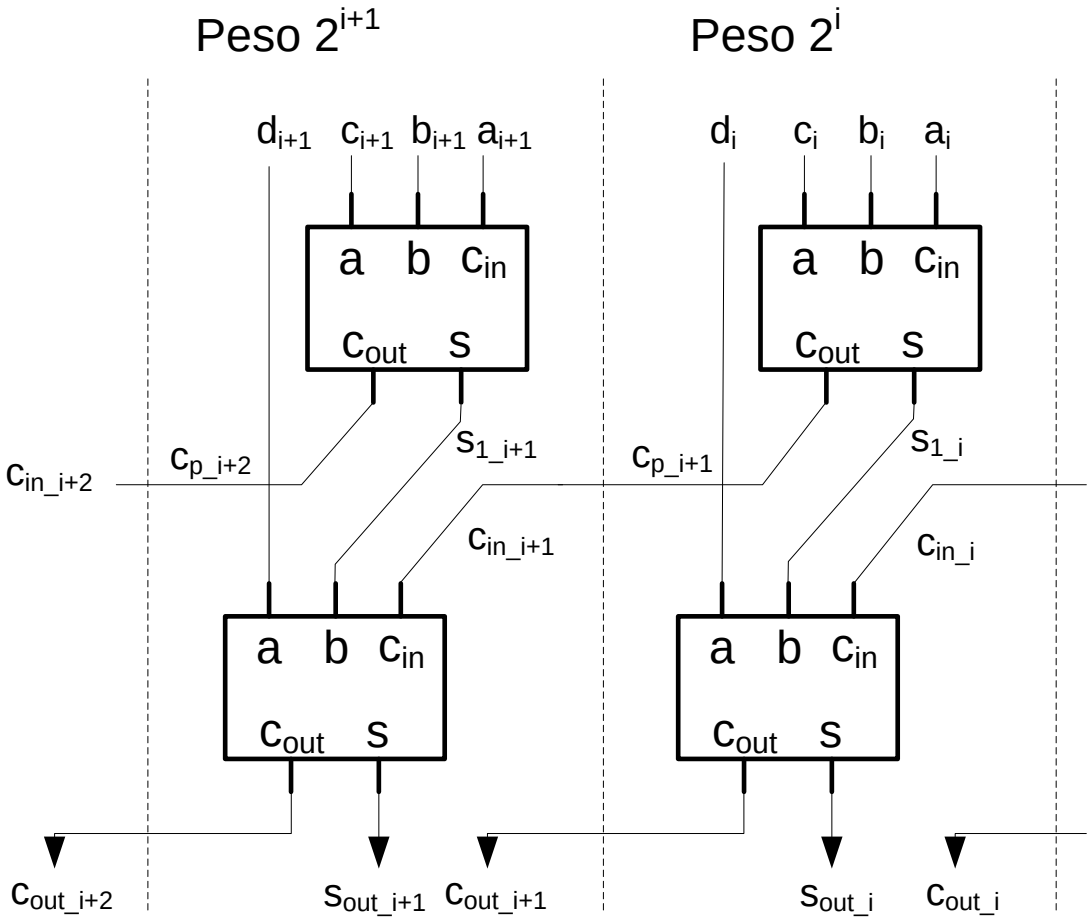


Figura 6-13 Compresor 4:2 a partir de compresores 3:2 (o FA)

Un sumador completo, mirado desde otro punto de vista, realiza la reducción de tres números binarios a dos números binarios, por lo que también se le llama compresor 3:2 o contador 3:2. Si se necesita realizar la suma de dos números expresados en formato carry–save, necesitaremos una reducción de 4 a 2, que se puede construir a partir de dos compresores 3:2 como se ha mostrado en la figura 6-13. En este caso el tiempo de cálculo para dos números CS de n-dígitos sería al equivalente a dos sumadores completos. En esta figura puede observarse como la propagación de acarreo se corta y como el acarreo propagado del primer nivel se debe conectar a la entrada de acarreo del segundo. Además se podría sumar un bit de m operandos y producir una palabra de suma S y otra de acarreo C, con lo que obtendríamos compresores 5:2 que realizan una reducción de cinco a dos bits, compresores 6:2 que realizarían una reducción de 6 bits de entrada a dos de salida y, en general, compresores p:t que realizarían la suma de p bits de entrada produciendo t bits de salida.

Por otro lado, los multiplicadores empotrados de que disponen la mayoría de las FPGAs actuales son relativamente pequeños, con un tamaño usual de 18x18 bits. Por lo

tanto, si queremos realizar la multiplicación de un mayor tamaño de operando, tenemos que combinar varios multiplicadores para la construcción de unidades de multiplicación superiores. Por ejemplo, para la realización de un multiplicador de 35x35 con signo utilizando multiplicadores empotrados de 18x18 bits con salida convencional, podemos hacer referencia a un estudio de Xilinx [41] (Pag. 381), que se describe en la figura 6-13. Aquí se muestra como se puede conseguir implementar un multiplicador con signo de 35x35 bits utilizando cuatro multiplicadores empotrados con signo y dos sumadores. El sumador realmente es de 53 bits de ancho, ya que los 17 bits menos significativos se obtienen directamente al multiplicar A[16:0] por B[16:0].

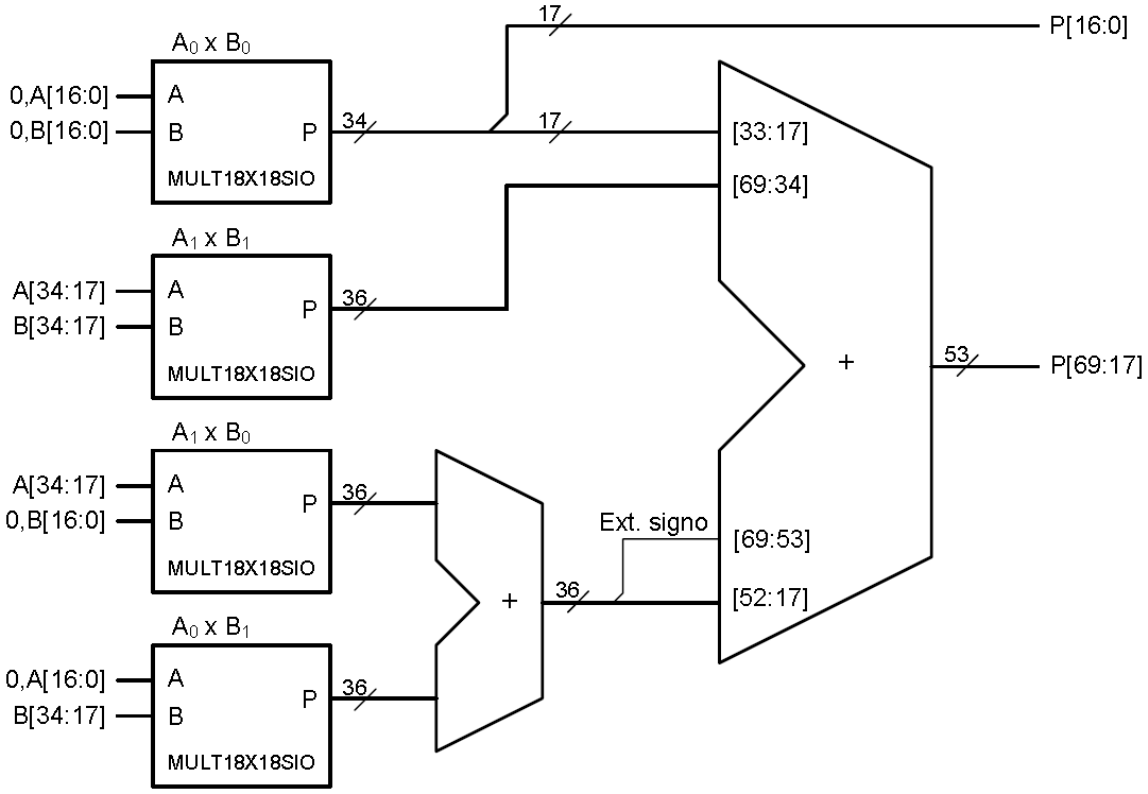


Figura 6-14 Multiplicador con signo 35x35 bits con salida convencional basado en bloques multiplicadores de 18x18

Los productos parciales que deben realizar los multiplicadores empotrados y las correspondientes sumas se describen en la figura 6-15:

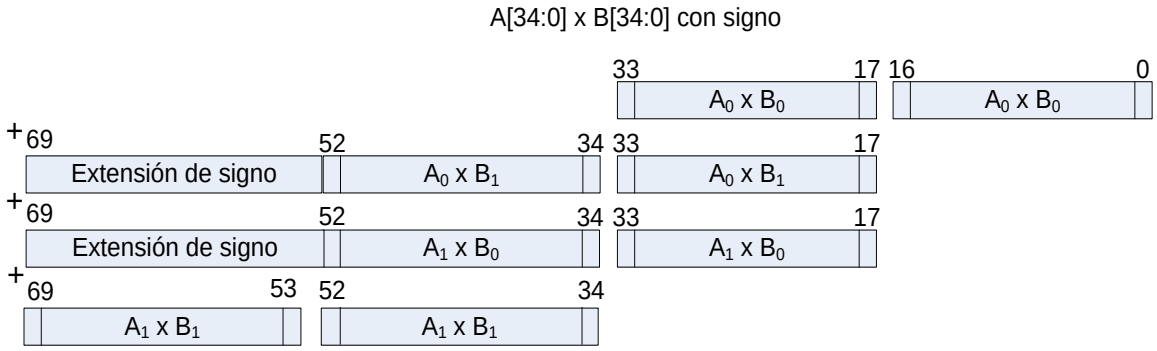


Figura 6-15 Productos parciales de un multiplicador de 35x35 con signo

La arquitectura de este multiplicador 35x35 con signo, utilizando los multiplicadores 18x18 de las FPGAs y sumadores completos (compresores 3:2), con la salida expresada en formato *carry–save* corresponderían a la de la figura 6-16.

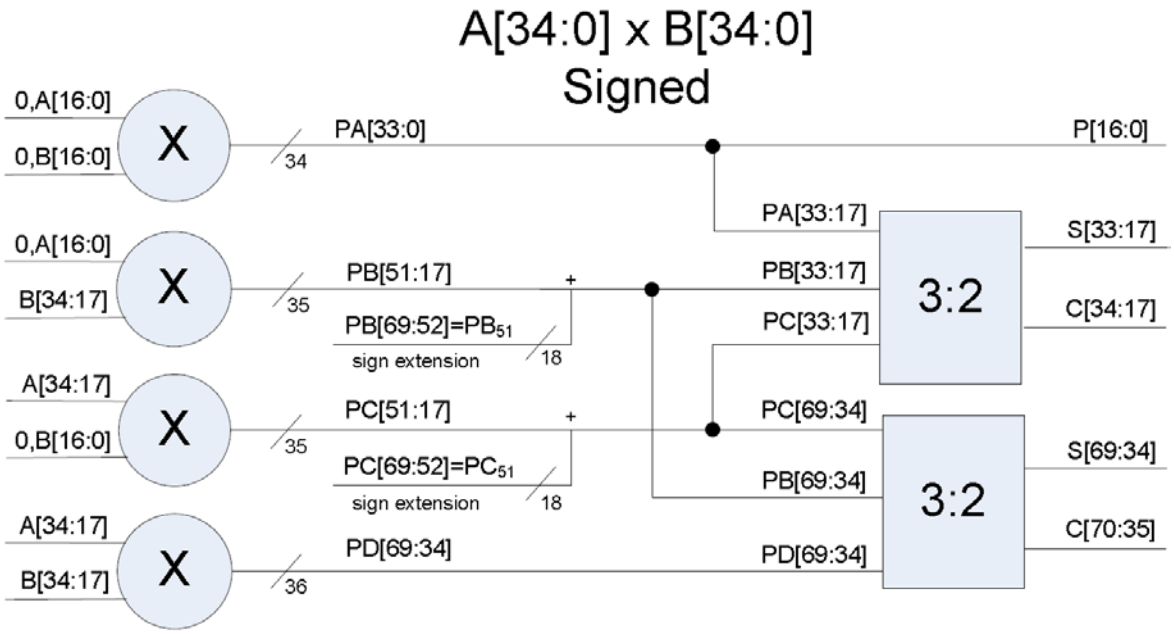


Figura 6-16 Arquitectura de un multiplicador 35x35 con signo con salida CSA basado en bloques multiplicadores de 18x18

6.4.1. Resultados obtenidos para la Spartan–3

Para obtener los resultados temporales que se emplean en la operación vamos a considerar que el tiempo total para la realización de una operación MAC (t_{opMAC}) dependerá del número de operaciones MAC ($n_{ciclos)}$ y el retardo en cada ciclo ($t_{retardociclo}$), es decir:

$$t_{opMAC} = n_{ciclos} * t_{retardociclo}$$

Para el caso de un MAC con salida convencional como en la figura 6-10 tendremos:

$$t_{opMAC} = n_{ciclos} * (t_{retardoMUL} + t_{retardoADD})$$

Para el caso de un MAC con salida en CSA como se describe en la figura 6-11:

$$t_{opMAC} = n_{ciclos} * (t_{retardoMUL} + t_{retardoCSA})$$

y en el caso que se desee obtener una salida convencional del MAC CSA, hay que sumar el retardo del sumador CPA final de dicha figura:

$$t_{opMAC} = n_{ciclos} * (t_{retardoMUL} + t_{retardoCSA}) + t_{retardoCPA}$$

En la tabla 6-5 se muestran los resultados obtenidos mediante la herramienta de síntesis de Xilinx ISE Design Suite 12.1 para una FPGA Spartan–3 XC3S-1600E de velocidad –5. Se tienen en cuenta dos valores distintos del ancho de los datos: para un MAC de ancho 35x35+12 bits con signo y de ancho 69x69+12 bits con signo. Se muestran los datos obtenidos para un MAC con salida convencional CPA descrito en VHDL con sumadores CPA, y para un MAC con multiplicadores CSA y sumador–acumulador CSA (todo en CSA).

Tabla 6-5. Resultados de la implementación del MAC para la Spartan–3

	MAC	Slice	LUTs	Mul 18x18	Retardo (ns)
35x35+12 bits	Descrito en VHDL con sumadores CPA, salida CPA	140	207	4	16,13
	Utilizando los multiplicadores CSA y sumador CSA (todo en CSA)	191	364	4	10,99
69x69+12 bits	Descrito en VHDL con sumador CPA, salida CPA	555	967	16	23,81
	Utilizando los multiplicadores CSA y sumador CSA (todo en CSA)	481	936	16	14,71

En el caso de que se desee una salida convencional para el MAC CSA hay que sumar a los resultados de la tabla 6-5, los resultados de la tabla 6-6 que corresponden a los obtenidos para el sumador CPA final que aparece en la figura 6-11.

Tabla 6-6. Retardo de sumadores CPA para la Spartan–3

CPA	Slice	LUTs	Retardo (ns)
35x35+12 bits	42	83	5,83
69x69+12 bits	76	151	9,34

- **Discusión de velocidad para la Spartan 3:**

Según los datos obtenidos en la tabla 6-5, para el caso en que el ancho de los datos sea de 35x35+12 bits, el tiempo de operación utilizado en la operación MAC descrito en VHDL con sumadores CPA y salida CPA se obtendría con la siguiente ecuación:

$$t_{opMAC} = n_{ciclos} * 16{,}13 \text{ ns}$$

y utilizando los multiplicadores descritos en CSA y con la salida convencional CPA se obtendrían a partir de la siguiente expresión:

$$t_{opMAC} = n_{ciclos} * 10{,}99 \text{ ns} + 5{,}83 \text{ ns}$$

Para el caso en que el ancho de los datos sea de 69x69+12 bits, el tiempo de operación invertido en la operación MAC descrito en VHDL con sumadores CPA y salida CPA se obtendría con la siguiente ecuación:

$$t_{opMAC} = n_{ciclos} * 23{,}81 \text{ ns}$$

y utilizando los multiplicadores descritos en CSA y con la salida convencional CPA se obtendrían a partir de la siguiente expresión:

$$t_{opMAC} = n_{ciclos} * 14{,}71 \text{ ns} + 9{,}34 \text{ ns}$$

Utilizando estos datos obtenemos la tabla 6-7 que nos indica de manera más explícita los resultados para distintos números de ciclos de operación (2, 4, 6, 10 y 20 ciclos).

Tabla 6-7. Retardos en la Spartan–3 en función del número de ciclos MAC

Spartan–3 XC3S1600E-5FG484 Grado 5		Número de ciclos. Retardo (ns)				
		2	4	6	10	20
35x35 + 12	MAC VHDL con salida convencional	32,26	64,52	96,78	161,3	322,6
	MAC CSA y salida CPA	27,81	49,79	71,77	115,73	225,63
69x69 + 12	MAC VHDL con salida convencional	47,62	95,24	142,86	238,1	476,2
	MAC CSA y salida CPA	38,76	68,18	97,6	156,44	303,54

Como se puede observar en la tabla 6-7, a partir de 2 ciclos, la arquitectura propuesta, aún con salida CPA, es más rápida que la descrita en VHDL. Por ejemplo, para el MAC 35*35 +12 y para 6 ciclos t_{opMAC} del MAC VHDL es 96,78 ns, que corresponde a una frecuencia de operación de 10,33 MopMAC/s, y en el caso del MAC CSA con salida CPA 71,77 ns, que corresponde a una frecuencia de operación de 13,93 MopMAC/s. En el caso de un número de ciclos elevado, el retardo del sumador CPA final en la arquitectura MAC CSA tiene una escasa influencia en el rendimiento, por tanto el rendimiento del MAC

realizado con aritmética redundante es de un 43% superior aproximadamente para el caso de 20 ciclos y 35x35+12.

Los datos obtenidos en la tabla 6-7 pueden observarse de una forma gráfica en la figura 6-17.

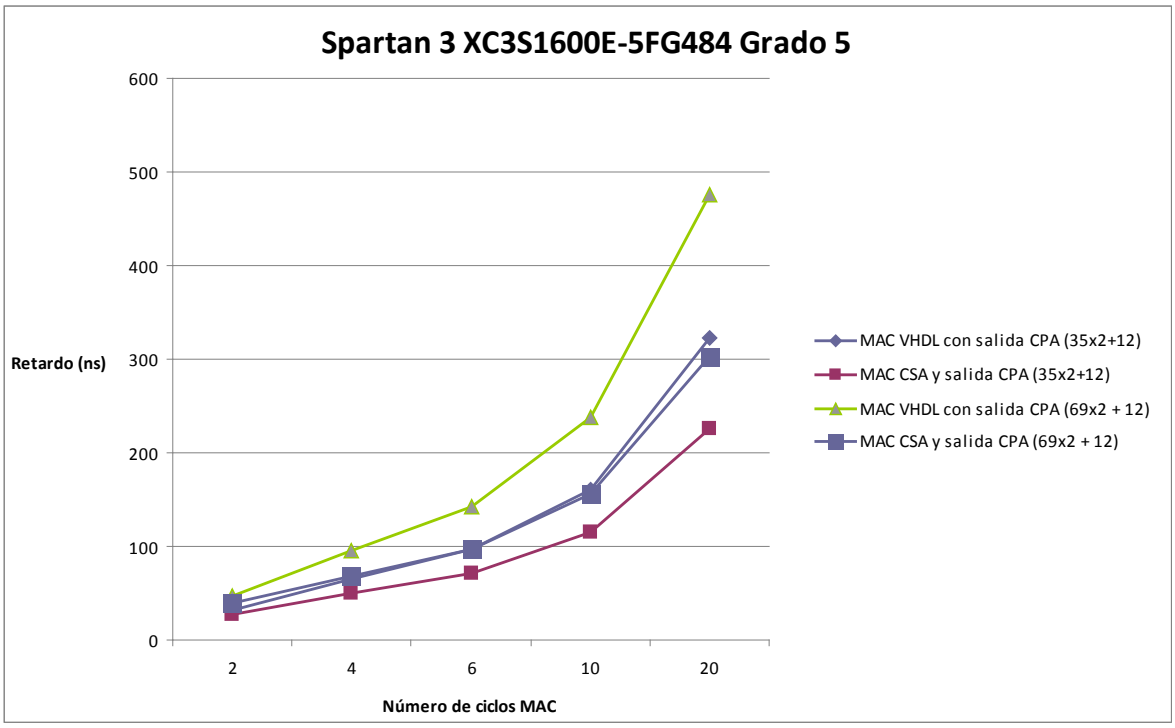


Figura 6-17 Retardos en la Spartan–3 del MAC

Otra característica destacable, es que el número de bits tiene una influencia en el retardo mucho menor en el caso de aritmética redundante que en el caso de aritmética normal. Esto es debido a que las sumas intermedias de los productos parciales en el multiplicador tienen un retardo similar para distintos anchos de bits en el caso CSA, mientras que en la descripción VHDL, al utilizarse sumadores CPA, tanto en los sumadores parciales del multiplicador como en el sumador final el retardo es proporcional al número de bits.

- **Discusión del área en el caso de la Spartan 3:**

En cuanto al área, el MAC CSA ocupa un área mayor, sin embargo esta diferencia, disminuye a medida que aumenta el ancho de las palabras, pasando de 1,37 veces superior en el caso de 35x35+12, a un área similar para el caso de 69x69+12.

En el caso de números sin signo, el área del multiplicador con aritmética redundante disminuye considerablemente así como la del sumador–acumulador, y la frecuencia de operación es similar en ambas arquitecturas.

6.4.2. Resultados obtenidos para la Spartan–6

La tabla 6-8 muestra los resultados para un MAC de ancho 35x35+12, con signo y de ancho 69x69+12, con signo para una FPGA Spartan–6 en concreto la Xilinx XC6SLX100–3FGG676 de velocidad –3. Se muestran los datos para un MAC con salida convencional descrito en VHDL con sumadores CPA, y para un MAC con multiplicadores y sumador–acumulador CSA.

Tabla 6-8. Resultados de la implementación del MAC para la Spartan–6

MAC		Slice	LUTs	DSP48	Retardo (ns)
35x35+12 bits	Descrito en VHDL con sumadores CPA, salida CPA	31	82	4	19,61
	Utilizando los multiplicadores CSA y sumador CSA (todo en CSA)	53	159	4	8,36
69x69+12 bits	Descrito en VHDL con sumador CPA, salida CPA	119	395	16	24,39
	Utilizando los multiplicadores CSA y sumador CSA (todo en CSA)	333	877	16	11,26

En el caso de que se desee una salida convencional para el MAC CSA hay que sumar a los resultados de la tabla 6-8, los resultados de la tabla 6-9 para el sumador CPA final según la arquitectura presentada en la figura 6-11, tal y como sucedía en el caso de la Spartan–3.

Tabla 6-9. Retardo de sumadores CPA para la Spartan–6

CPA	Slice	LUTs	Retardo (ns)
35x35+12 bits	21	82	3,97
69x69+12 bits	38	150	5,41

- **Discusión de velocidad para la Spartan 6:**

Aún en el caso de pedir al sintetizador el máximo esfuerzo en velocidad, la herramienta de síntesis Xilinx ISE Design Suite 12.1, siempre aprovecha los sumadores incluidos en los bloques DSP. Esto se encuentra detallado y justificado en el documento de Xilinx [44] donde se explican los módulos DSP48A1 de esta FPGA y una explicación de la síntesis.

Esta forma de sintetizar hace que el área ocupada sea muy pequeña pero produce un encadenamiento en los bloques DSP que elevan el retardo.

Según los datos obtenidos en la tabla 6-8, para el caso en que el ancho de los datos sea de 35x35+12 bits, el tiempo de operación consumido en la operación MAC descrito en VHDL con sumadores CPA y salida CPA se obtendría con la siguiente ecuación:

$$t_{opMAC} = n_{ciclos} * 19{,}61 \text{ ns}$$

y utilizando los multiplicadores descritos en CSA y con la salida convencional CPA se obtendrían a partir de la siguiente expresión:

$$t_{opMAC} = n_{ciclos} * 8{,}36 \text{ ns} + 3{,}97 \text{ ns}$$

Para el caso en que el ancho de los datos sea de 69x69+12 bits, el tiempo de operación consumido en la operación MAC descrito en VHDL con sumadores CPA y salida CPA se obtendría con la siguiente ecuación:

$$t_{opMAC} = n_{ciclos} * 24{,}39 \text{ ns}$$

y utilizando los multiplicadores descritos en CSA y con la salida convencional CPA se obtendrían a partir de la siguiente expresión:

$$t_{opMAC} = n_{ciclos} * 11{,}26 \text{ ns} + 5{,}41 \text{ ns}$$

Utilizando estas ecuaciones y los datos experimentales de la tabla 6-9, obtenemos la tabla 6-10 que nos indica de manera más clarificante los resultados para distintos números de ciclos de operación (2, 4, 6, 10 y 20 ciclos).

Tabla 6-10. Retardos en la Spartan–6 en función del número de ciclos MAC

Spartan 6 XC6SLX100-3FGG676		Número de ciclos. Retardo (ns)				
		2	4	6	10	20
35x35 + 12	MAC VHDL con salida convencional	39,22	78,44	117,66	196,1	392,2
	MAC CSA y salida CPA	20,69	37,41	54,13	87,57	171,17
69x69 + 12	MAC VHDL con salida convencional	48,78	97,56	146,34	243,9	487,8
	MAC CSA y salida CPA	27,93	50,45	72,97	118,01	230,61

En la figura 6-18 podemos observar gráficamente estos resultados de forma más visual.

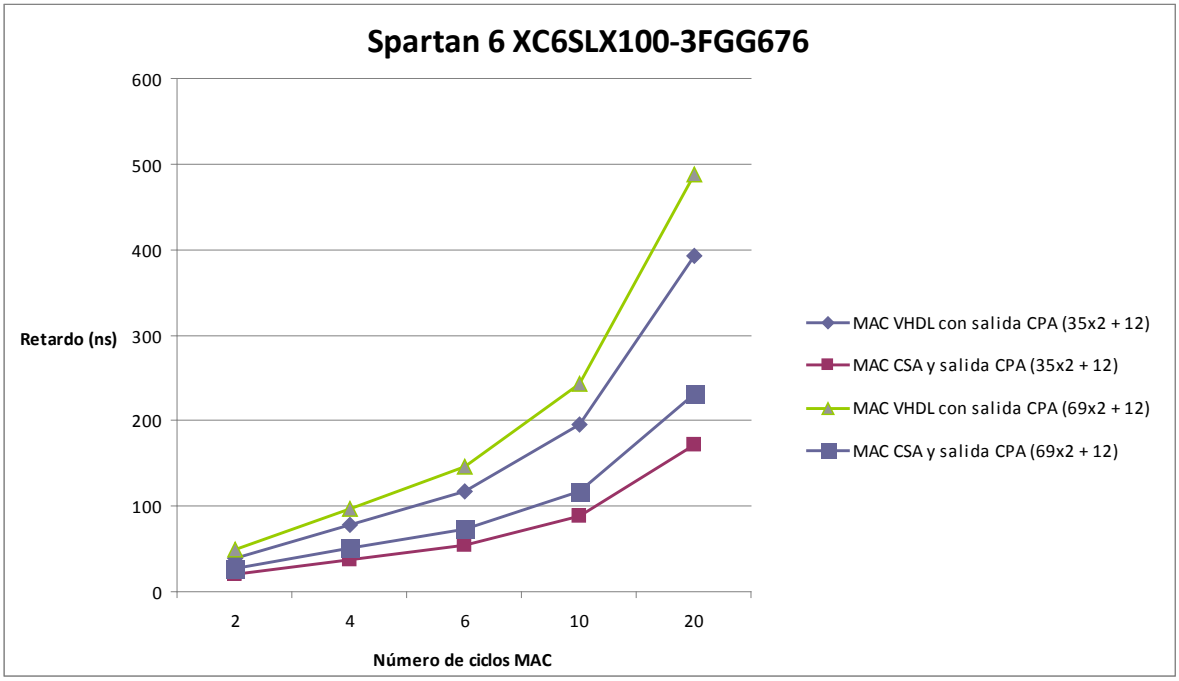


Figura 6-18 Retardos en la Spartan–6 del MAC

Como se puede observar en la tabla 6-10 al igual que en el caso de la Spartan–3, a partir de 2 ciclos la arquitectura propuesta, aún con salida CPA, es más rápida que la descrita en VHDL. Por ejemplo, para el MAC 35*35 +12 y para 6 ciclos t_{opMAC} VHDL es 117,66 ns. que corresponde a una frecuencia de operación de 8,50 MopMAC/s (Millones de operaciones MAC por segundo) y en el caso del MAC CSA con salida CPA 54,13 ns que corresponde a una frecuencia de operación de 18,47 MopMAC/s. En el caso de un número de ciclos elevado el retardo del sumador CPA final en la arquitectura MAC CSA tiene una escasa influencia en el rendimiento, por tanto el rendimiento del MAC realizado con aritmética redundante es de 2,17 veces superior, más del doble.

Otra característica a destacar, es que en el tiempo de retardo del MAC implementado con aritmética redundante, el número de bits tiene una escasa influencia. Esto es debido a que las sumas intermedias de los productos parciales en el multiplicador tienen un retardo similar en el caso CSA, mientras que en la descripción VHDL, al utilizarse sumadores CPA, tanto en los sumadores parciales del multiplicador como en el sumador final el retardo va aumentando conforme al tamaño de los operandos.

- **Discusión del área en el caso de la Spartan 6:**

En cuanto al área, el MAC CSA ocupa un área mucho mayor, sin embargo esta diferencia disminuye a medida que aumenta el ancho de las palabras pasando de 4 veces el caso de

35x35+12 superior a 3,2 veces para el caso de 69x69+12. Es una diferencia considerable pero si tenemos en cuenta el retardo, la disminución de área por la utilización de sumadores en cascada en la implementación VHDL aumenta enormemente el retardo.

Al igual que en la Spartan–3 en el caso de números sin signo, el área del multiplicador con aritmética redundante disminuye considerablemente así como la del sumador–acumulador, y la frecuencia de operación es similar en ambas arquitecturas.

6.5. Una arquitectura optimizada en velocidad para la implementación del MAC en LUT-6 utilizando compresores de salida de doble acarreo

Una optimización del MAC en velocidad se puede conseguir utilizando compresores de doble acarreo. Del estudio de las LUT6 de la Spartan–6, se observa que se puede conseguir un compresor 5:3, con un retardo menor que un compresor 4:2 genérico y un consumo de área similar como se explica en el *report* interno de nuestro grupo de investigación de Ortiz et al. [95]. Para un solo bit, el consumo en LUT es el mismo para un compresor 5:3 que un compresor 4:2. Sin embargo, si se consideran más bits, se puede optimizar el compresor 4:2 para que se consuman 3 LUT por cada dos bits por lo que el compresor 5:3 ocupa un área mayor, pero con un retardo menor. Unido a esto, está la facilidad de implementación de sumadores ternarios en la Spartan–6, lo que permiten la suma final de tres operandos con un consumo análogo en área. En cuanto a velocidad los sumadores ternarios son más lentos, pero como se ha comentado ampliamente en los estudios de los capitulos anteriores, la velocidad del sumador final influye muy poco, en el total de tiempo utilizado para la operación MAC.

La tabla 6-11 muestra los resultados obtenidos que comparan un sumador de 2 y de 3 operandos en cuanto a consumo de espacio y velocidad.

Tabla 6-11. Comparación de un sumador de 2 operandos frente al de 3 operandos

Spartan 6	Sumador (A+B)-CPA			Sumador (A+B+C)-Ternario		
Anchura del sumador	Slice	LUT	Retardo (ns)	Slice	LUT	Retardo (ns)
35x35 + 12	21	82	3,97	21	84	4,79
69x69 + 12	38	150	5,41	38	151	6,37

De estos datos se observa que el área consumida es del mismo orden y el retardo es ligeramente mayor. De ahí que se plantee la necesidad de estudiar la implementación del MAC utilizando compresores de salida de doble acarreo, teniendo multiplicadores CSA y

teniendo multiplicadores de salida de doble acarreo. Se puede implementar un MAC utilizando un multiplicador con salida CSA y un sumador–acumulador con compresores de doble acarreo de salida 5:3 y un sumador ternario final. La otra opción es utilizar un multiplicador con salida de doble acarreo y un sumador–acumulador de salida de doble acarreo, y para la suma final utilizar igualmente un sumador ternario.

6.5.1. MAC en LUT 6 utilizando un multiplicador con salida CSA y un acumulador con salida de doble acarreo utilizando compresores 5:3

La figura 6-19 muestra la estructura de un MAC implementado con multiplicadores con salida CSA, compresores 5:3 en el sumador–acumulador y con un sumador ternario para obtener una salida convencional.

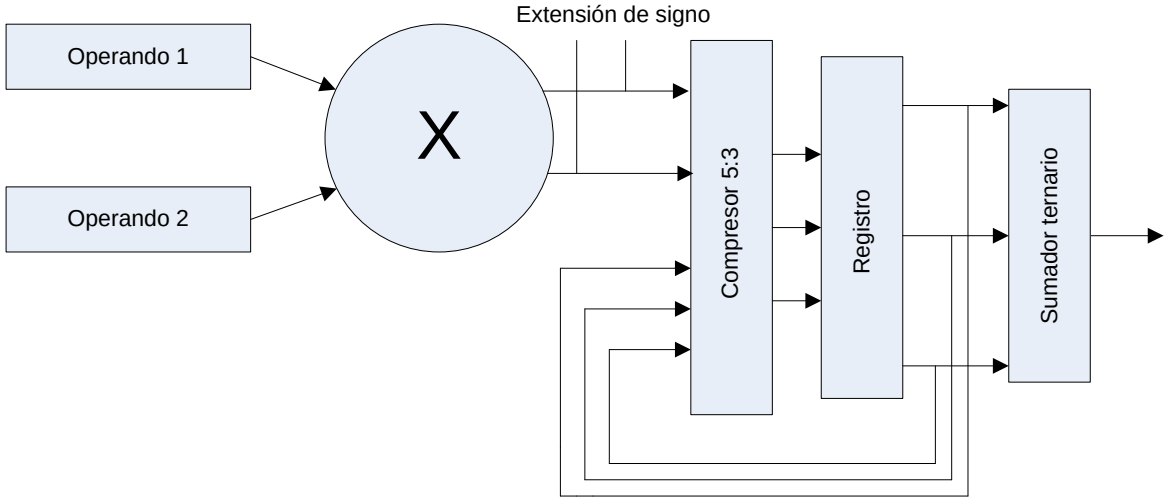


Figura 6-19 MAC utilizando compresores 5:3 para el sumador–acumulador

La tabla 6-12 muestra los resultados de implementación para una Spartan–6 utilizando multiplicadores con salida CSA y un compresor 5:3 como sumador–acumulador. Se muestra también el resultado del MAC descrito en VHDL utilizado en este trabajo para los resultados comparativos que obviamente serán los mismos que los de las tablas del apartado anterior. Como en los apartados anteriores, se muestran los resultados que se obtienen para los anchos de palabra de 35x35+12 y 69x69+12.

Tabla 6-12. Resultados de la implementación del MAC para la Spartan–6 utilizando compresores 5:3

MAC	Slice	LUTs	DSP48A1	Retardo (ns)

35x35+12 bits	Descrito en VHDL con sumadores CPA, salida CPA	31	82	4	19,61
	Utilizando los multiplicadores CSA y sumador CSA de doble acarreo	70	230	4	8,17
69x69+12 bits	Descrito en VHDL con sumador CPA, salida CPA	119	395	16	24,39
	Utilizando los multiplicadores CSA y sumador CSA de doble acarreo	323	818	16	10,61

- **Discusión de velocidad para la Spartan 6:**

Según los datos obtenidos en la tabla 6-12, para el caso en que el ancho de los datos sea de 35x35+12 bits, el tiempo de operación consumido en la operación MAC descrito en VHDL con sumadores CPA y salida CPA se obtendría con la siguiente ecuación:

$$t_{opMAC} = n_{ciclos} * 19{,}61 \text{ ns}$$

y utilizando los multiplicadores con salida en carry–save, compresores 5:3 como sumador–acumulador de salida de doble acarreo e incluyendo el retardo del sumador CPA ternario final para obtener la salida convencional CPA de la tabla 6-11, se calcularían a partir de la siguiente expresión:

$$t_{opMAC} = n_{ciclos} * 8{,}17 \text{ ns} + 4{,}79 \text{ ns}$$

Para el caso en que el ancho de los datos sea de 69x69+12 bits, el tiempo de operación consumido en la operación MAC descrito en VHDL con sumadores CPA y salida CPA se obtendría con la siguiente ecuación:

$$t_{opMAC} = n_{ciclos} * 24{,}39 \text{ ns}$$

y utilizando los multiplicadores descritos con salida carry–save, con sumador–acumulador de salida de doble acarreo (compresor 5:3) incluyendo el retardo del sumador ternario final para conseguir una salida convencional CPA (tabla 6-11) se obtienen con la siguiente expresión:

$$t_{opMAC} = n_{ciclos} * 10{,}61 \text{ ns} + 6{,}37 \text{ ns}$$

Utilizando estas ecuaciones obtenemos la tabla 6-13 indicando los resultados para distintos números de ciclos de operación (2, 4, 6, 10 y 20 ciclos).

Tabla 6-13. Retardos en MAC en la Spartan–6 con acumulador con compresor 5:3

Spartan 6 XC6SLX100-3FGG676		Número de ciclos. Retardo (ns)				
		2	4	6	10	20
35x35+12	MAC VHDL con salida CPA	39,22	78,44	117,66	196,1	392,2
	MAC CSA - Mul CSA sumador de doble acarreo y salida CPA	21,13	37,47	53,81	86,49	168,19
69x69+12	MAC VHDL con salida CPA	48,78	97,56	146,34	243,9	487,8
	MAC CSA - Mul CSA sumador de doble acarreo y salida CPA	27,59	48,81	70,03	112,47	218,57

En la figura 6-20 representamos estos mismos datos para realizar una comparativa más visual.

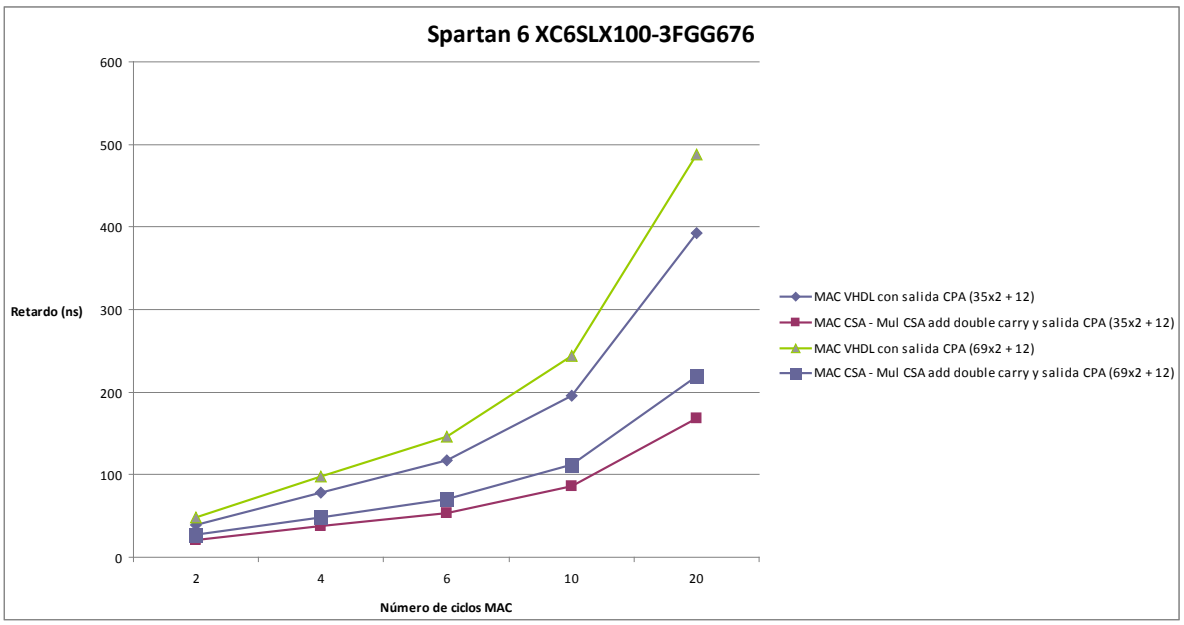


Figura 6-20 Retardos en MAC de la Spartan–6, sumador–acumulador con compresores 5:3

Como se puede observar, igual que en el caso anterior de aritmética redundante, utilizando en parte la implementación aritmética redundante de doble acarreo, a partir de 2 ciclos la arquitectura propuesta, aún con salida CPA, es más rápida que la descrita en VHDL. Por ejemplo, para el MAC 35*35 +12 y para 6 ciclos t_{opMAC} VHDL es 117,66 ns, que corresponde a una frecuencia de operación de 8,50 MopMAC/s, y en el caso del MAC CSA con compresores 5:3 y con salida CPA 53,81 ns, que corresponde a una frecuencia de operación de 18,58 MopMAC/s.

En el caso de un número de ciclos elevado el retardo del sumador CPA final en la arquitectura MAC CSA tiene una escasa influencia en el rendimiento, por tanto el rendimiento del MAC realizado con aritmética redundante es de un 2,19 veces superior.

Otra característica destacable, es que en el tiempo de retardo del MAC implementado con aritmética redundante, el número de bits tiene una escasa influencia. Esto es debido a que las sumas intermedias de los productos parciales en el multiplicador tienen un retardo similar en el caso CSA, mientras que en la descripción VHDL, al utilizarse sumadores CPA, tanto en los sumadores parciales del multiplicador como en el sumador final, el retardo es proporcional al número de bits.

- **Discusión del área en el caso de la Spartan 6:**

En cuanto al área, el MAC CSA ocupa un área mucho mayor, sin embargo disminuye a medida que aumenta el ancho de las palabras pasando de 2,80 veces superior para caso de 35x35+12, a el doble para el caso de 69x69+12.

6.5.2. MAC en LUT 6 utilizando un multiplicador con salida CSA de doble acarreo y un acumulador con salida de doble acarreo utilizando compresores 6:3

La figura 6-21 muestra la estructura de un MAC implementado con multiplicadores con salida CSA de doble acarreo, compresores 6:3 y con un sumador ternario final, al igual que en el caso anterior, para obtener una salida convencional.

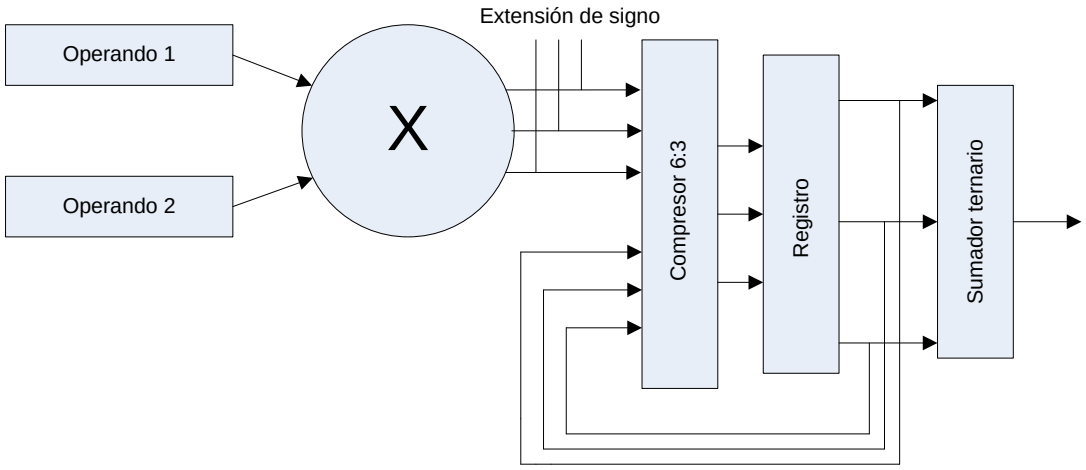


Figura 6-21 MAC utilizando compresores 6:3

La tabla 6-14 muestra los resultados de implementación para una Spartan–6 utilizando multiplicadores con salida CSA de doble acarreo, compresores 6:3 como acumulador que proporciona el resultado de doble acarreo y un sumador ternario final. Se muestra también, con objeto de comparar los resultados, el obtenido con un MAC descrito en VHDL y salida CPA.

Tabla 6-14. Resultados de la implementación del MAC para la Spartan–6 utilizando compresores 6:3

MAC		Slice	LUTs	DSP48A1	Retardo (ns)
35x35+12	Descrito en VHDL con sumadores CPA, salida CPA	31	82	4	19,61
	Con multiplicadores CSA de doble acarreo y sumador CSA de doble acarreo	61	198	4	7,2
69x69+12	Descrito en VHDL con sumador CPA, salida CPA	119	395	16	24,39
	Con multiplicadores CSA de doble acarreo y sumador CSA de doble acarreo	407	919	16	9,21

- **Discusión de velocidad para la Spartan 6:**

Según los datos obtenidos en la tabla 6-14, para el caso en que el ancho de los datos sea de 35x35+12 bits, el tiempo de operación consumido en la operación MAC descrito en VHDL con sumadores CPA y salida CPA se obtendría con la siguiente ecuación:

$$t_{opMAC} = n_{ciclos} * 19{,}61 \text{ ns}$$

y utilizando los multiplicadores con salida en *carry–save*, compresores 6:3 como sumador–acumulador de salida de doble acarreo e incluyendo el retardo del sumador CPA ternario final para obtener la salida convencional CPA se calcularían a partir de la siguiente expresión:

$$t_{opMAC} = n_{ciclos} * 7{,}2 \text{ ns} + 4{,}79 \text{ ns}$$

Para el caso en que el ancho de los datos sea de 69x69+12 bits, el tiempo de operación consumido en la operación MAC descrito en VHDL con sumadores CPA y salida CPA se obtendría con la siguiente ecuación:

$$t_{opMAC} = n_{ciclos} * 24{,}39 \text{ ns}$$

y utilizando los multiplicadores descritos con salida *carry–save*, con sumador–acumulador de salida de doble acarreo (compresor 6:3) incluyendo el retardo del sumador ternario final para conseguir una salida convencional CPA se obtienen con la siguiente expresión:

$$t_{opMAC} = n_{ciclos} * 9{,}21 \text{ ns} + 6{,}37 \text{ ns}$$

Estas ecuaciones junto con los datos de la tabla 6-14 producen los resultados expuestos en la tabla 6-15 que indica los valores para distintos números de ciclos de operación MAC (2, 4, 6, 10 y 20 ciclos).

Tabla 6-15. Retardos en MAC en la Spartan–6 con aritmética de doble acarreo y compresor 6:3

Spartan 6 XC6SLX100-3FGG676		Número de ciclos. Retardo (ns)				
		2	4	6	10	20
35x2+ 12	MAC VHDL con salida CPA	39,22	78,44	117,66	196,1	392,2
	MAC CSA doble acarreo y salida CPA	19,19	33,59	47,99	76,79	148,79
69x2 + 12	MAC VHDL con salida CPA	48,78	97,56	146,34	243,9	487,8
	MAC CSA doble acarreo y salida CPA	24,79	43,21	61,63	98,47	190,57

En la figura 6-22 representamos estos mismos datos para obtener gráficamente estos resultados comparativos.

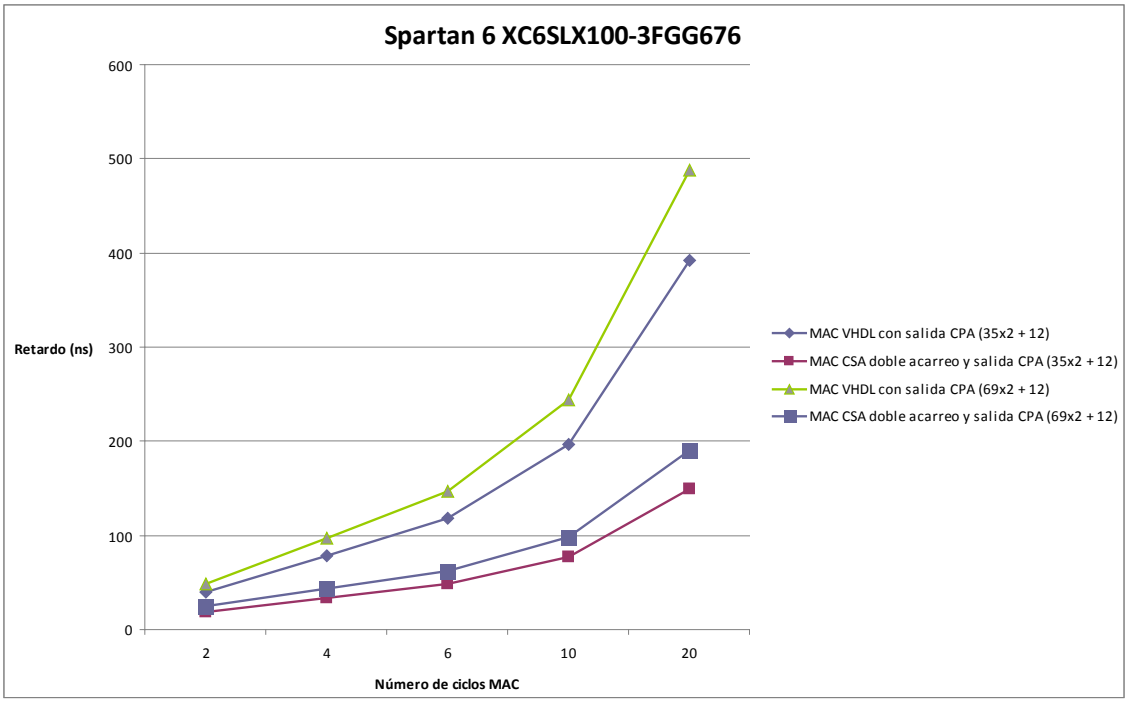


Figura 6-22 Retardos en MAC en la Spartan–6, aritmética CSA de doble acarreo

Como se puede observar, igual que en el caso anterior de aritmética redundante, utilizando implementación aritmética redundante de doble acarreo, a partir de 2 ciclos la arquitectura propuesta, aún con salida CPA, es más rápida que la descrita en VHDL. Por ejemplo, para el MAC 35*35 +12 y para 6 ciclos t_{opmac} VHDL es 117,66 ns, que corresponde a una frecuencia de operación de 8,50 MopMAC/s, y en el caso del MAC CSA con compresores 6:3 en el sumador–acumulador y con salida CPA el retardo obtenido es de 47,99 ns, correspondiente a una frecuencia de 20,84 MopMAC/s.

En el caso de un número de ciclos elevado el retardo del sumador CPA final en la arquitectura MAC CSA tiene una escasa influencia en el rendimiento, por tanto el rendimiento del MAC realizado con aritmética redundante es de un 2,45 veces superior.

Otra característica destacable, es que en el tiempo de retardo del MAC implementado con aritmética redundante, el número de bits tiene una escasa influencia. Esto es debido a que las sumas intermedias de los productos parciales en el multiplicador tienen un retardo similar en el caso CSA, mientras que en la descripción VHDL, al utilizarse sumadores CPA, tanto en los sumadores parciales del multiplicador como en el sumador final, el retardo aumenta conforme aumenta el número de bits de los operandos.

- **Discusión del área en el caso de la Spartan 6:**

En cuanto al área, el MAC CSA ocupa un área mucho mayor, sin embargo disminuye a medida que aumenta el ancho de las palabras pasando de 2,41 veces superior en el caso de 35x35+12, a 2,33 veces para el caso de 69x69+12.

6.6. Comparación de resultados y conclusiones

En los apartados anteriores se ha presentado la arquitectura de un MAC realizado con aritmética normal descrito en VHDL y realizada con aritmética redundante CSA y aritmética de doble acarreo. Como se ha comprobado la utilización de aritmética redundante aumenta enormemente el rendimiento y se ha comparado una a una las arquitecturas presentadas. En este apartado se pretende realizar una comparación entre las cuatro arquitecturas presentadas. En primer lugar se realiza la comparativa para el ancho de palabra 35x35+12 y a continuación con los operandos de tamañlo 69x69+12.

La tabla 6-16 y la figura 6-23 muestra los resultados para un MAC de ancho 35x35+12 bits. Como se puede observar la utilización de una MAC implementado con aritmética de doble acarreo en toda la implementación, tanto en el multiplicador como en el sumador–acumulador, es la que presenta un retardo menor y por tanto un mayor rendimiento.

Tabla 6-16. Retardos en MAC en la Spartan–6 para el caso de 35x35+12 bits

Spartan 6 XC6SLX100-3FGG676	Número de ciclos. Retardo (ns)				
	2	4	6	10	20

35x2+ 12	MAC VHDL con salida CPA	39,22	78,44	117,66	196,1	392,2
	MAC CSA y salida CPA Compresor 4:2	20,69	37,41	54,13	87,57	171,17
	Multiplicadores CSA y sumador CSA de doble acarreo. Compresor 5:3	21,13	37,47	53,81	86,49	168,19
	Multiplicadores CSA de doble acarreo y sumador CSA de doble acarreo. Compresor 6:3	19,19	33,59	47,99	76,79	148,79

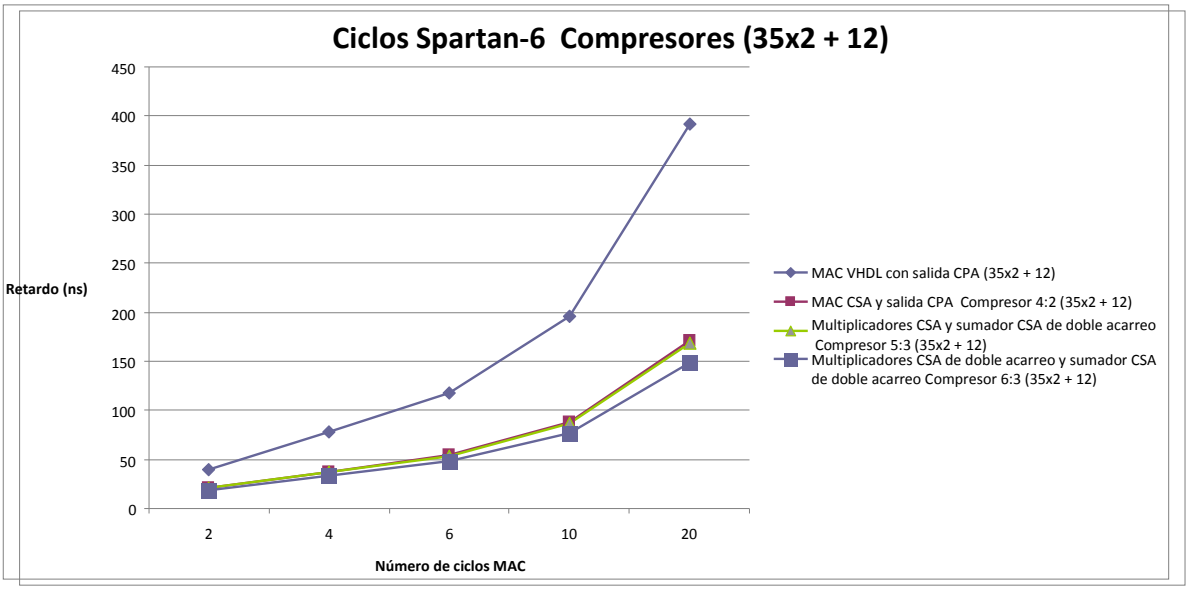


Figura 6-23 Retardos del MAC en la Spartan–6 para el caso de 35x35+12 bits

En la tabla 6-17 y la figura 6-24 se presentan los resultados para los 4 tipos de MAC considerados de ancho 69x69+12 bits .

Tabla 6-17. Retardos en MAC en la Spartan–6 para el caso de 69x69+12 bits

Spartan 6 XC6SLX100-3FGG676		Número de ciclos. Retardo (ns)				
		2	4	6	10	20
69x2+ 12	MAC VHDL con salida CPA	48,78	97,56	146,34	243,9	487,8
	MAC CSA y salida CPA Compresor 4:2	27,93	50,45	72,97	118,01	230,61
	Multiplicadores CSA y sumador CSA de doble acarreo. Compresor 5:3	27,59	48,81	70,03	112,47	218,57
	Multiplicadores CSA de doble acarreo y sumador CSA de doble acarreo. Compresor 6:3	24,79	43,21	61,63	98,47	190,57

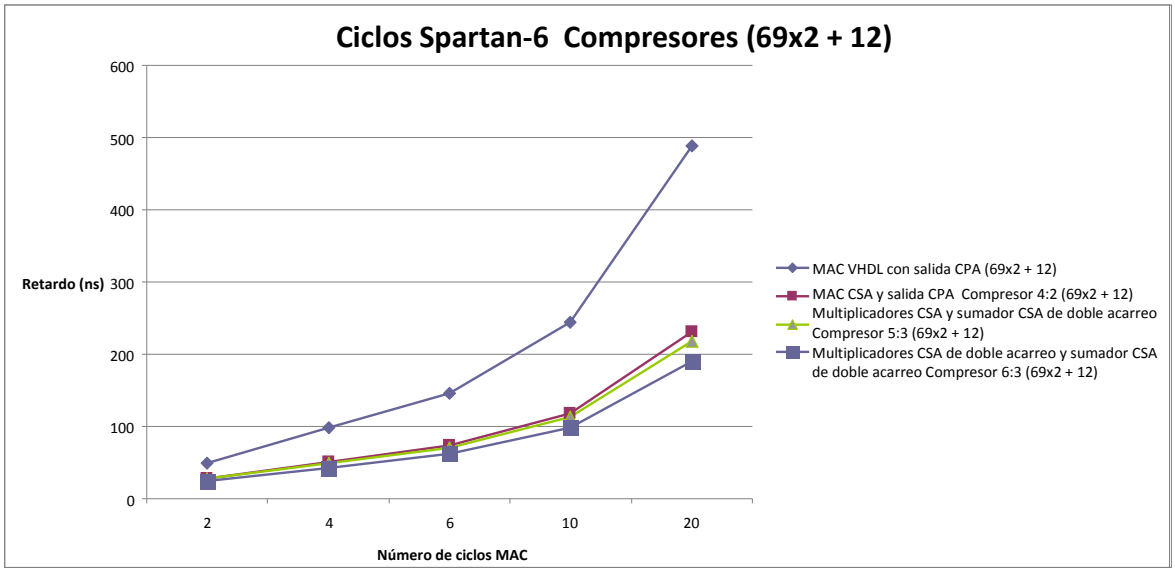


Figura 6-24 Retardos del MAC en la Spartan–6 para el caso de 69x69+12 bits

En cuanto a consumo de área la tabla 6-18 muestra los resultados tanto para un MAC de 35x35+12 como para un MAC de 69x69+12 de LUT para los 4 casos comentados en el párrafo anterior.

Tabla 6-18. Ocupación del MAC en la Spartan 6

Spartan 6 XC6SLX100-3FGG676	35x2 + 12		69x2 + 12	
	Slice	LUT	Slice	LUT
MAC VHDL con salida CPA	31	82	119	395
MAC CSA y salida CPA Compresor 4:2	53	159	333	877
Multiplicadores CSA y sumador CSA de doble acarreo. Compresor 5:3	70	230	323	818
Multiplicadores CSA de doble acarreo y sumador CSA de doble acarreo. Compresor 6:3	61	198	407	919

Puede observarse en esta tabla que el área consumida es menor en la descripción VHDL con sumadores CPA, pero como se ha visto anteriormente, lo que en realidad se está optimizando es el retardo que es mucho mayor para esta descripción. Por otro lado, se observa en estos datos que el área consumida para la implementación del tamaño 35x35+12 es mayor en el caso de la arquitectura con compresores 5:3 que para el caso de la arquitectura con compresores 6:3. Esto podemos justificarlo indicando que en el caso de la arquitectura de la figura 6-19 la herramienta de síntesis utiliza LUT para almacenar los resultados intermedios de cada operación, porque los multiplicadores no tienen salida de doble acarreo. Es decir para multiplicadores 35 x 35 (ver Fig. 6-15) la herramienta de síntesis utiliza la misma LUT para realizar el producto con salida de doble acarreo que para la acumulación para la arquitectura con multiplicador de salida CSA de doble acarreo y acumulador también con salida de doble acarreo (Fig. 6-21), en cambio para la arquitectura

con multiplicador CSA y acumulador con salida de doble acarreo (compresores 5:3, Fig. 6-19) la herramienta de síntesis utiliza una LUT para la salida del multiplicador y otra para el compresor 5:3.

En conclusión, podemos afirmar que la arquitectura más optimizada en la Spartan 6 corresponde a la última (compresores 6:3), ya que se obtienen los mejores retardos para la operación MAC cuando el número de ciclos es elevado, como es el caso que necesitamos para el cálculo de la convolución. El consumo en área no representa un consumo excesivo y además mejora al caso de la arquitectura con compresores 5:3.

7 Comparación de la operación de convolución en FPGA, ARM y DSP

RESUMEN

En este capítulo se describe una comparación de la operación de convolución en tres tipos de dispositivos de bajo coste: las FPGAs utilizadas en el capítulo anterior (Spartan 3 y Spartan-6), los procesadores digitales de señal (TMS320C6713 de Texas Instrument) y dos procesadores de propósito general de la tecnología ARM (el clásico ARM7 y el más actual Cortex–M3.

7.1. Introducción

La convolución es una operación costosa desde el punto de vista computacional, y es demasiado lenta cuando se implementa en un microprocesador. La posible necesidad de la reconfigurabilidad impide soluciones hardware dedicadas (ASIC) lo cual deja como opción soluciones basadas en procesadores de propósito general (GPPs), procesadores digitales de señal (DSPs), unidades de procesamiento gráfico (GPU) y las FPGAs. Las GPUs y los procesadores de propósito general tienden a consumir demasiada potencia para ser viables en aplicaciones de bajo coste y generalmente necesitan demasiados recursos lógicos para interconectarse con el resto del sistema. Los DSPs, las FPGAs y otros procesadores para aplicaciones empotradas, sin embargo disipan menos potencia obteniendo un mayor rendimiento, y normalmente disponen de una lógica que permiten la conexión directa con otros componentes del sistema.

En este capítulo comparo los resultados obtenidos para el cálculo de la operación de convolución en dispositivos para aplicaciones empotradas tales como dos FPGA de Xilinx Spartan–3 y Spartan–6, en un DSP de coma flotante de Texas Instrument TMS320C6713, y en dos procesadores de propósito general de la familia ARM como el ARM7 y el ARM Cortex M3, como ejemplos representativos de circuitos electrónicos de bajo coste. El objetivo de esta comparación es realizar un análisis cuantitativo de los tiempos que se

tardaría en realizar la operación de convolución en cada uno de estos dispositivos, ya que a priori se supone que los mejores tiempos lo daría la implementación en una FPGA porque optimizará mejor el hardware, seguido del DSPs al ser un procesador especializado en la operación MAC (multiplica y acumula) y por último las dos opciones de ARM.

La arquitectura de las FPGAs de Xilinx Spartan 3 y Spartan 6 ya quedó estudiada en el capítulo 4 de esta tesis, así como la arquitectura general de los procesadores digitales de señal (DSPs) por lo cual en este capítulo voy a describir detalladamente las arquitecturas de los DSPs de Texas Instrument y de los microcontroladores ARM. En particular me centraré en la arquitectura de los dispositivos en que se realizan las resultados experimentales: el DSP TMS320C6713 de Texas Instrument y en los dos procesadores ARM7 y ARM Cortex M3.

Los resultados de este capítulo fueron publicados en una ponencia en el III Congreso Científico de Investigadores en Formación de la Universidad de Córdoba celebrado en Córdoba los días 9 y 10 de Abril de 2013, organizado por la Escuela Internacional de Doctorado del Campus de Excelencia Internacional en Agroalimentación (ceiA3, eidA3) y Escuela Multidisciplinar de Doctorado de la UCO (ED-UCO). En este congreso se presentó la comunicación oral que lleva por título "*Comparación de la operación de convolución en diferentes dispositivos electrónicos de bajo coste*" [96]

La comparación de distintos dispositivos ha sido estudiada en diferentes ocasiones. Podemos señalar la comparación de las operaciones de procesamiento digital de señales entre procesadores DSPs y FPGAs en [97], [98] y [99], entre CPUs y FPGAs en [100] y [101], y otros artículos añaden las GPUs tales como en [102], [103] y[104] .

7.2. Familia TMS320 DSP de Texas Instruments

En 1982 Texas Instrument introdujo en el mercado su primer procesador de señal programable de uso general (DSP), cuya denominación fue TMS32010 DSP. Este era capaz de operar cinco millones de instrucciones por segundo. Sus principales aplicaciones estuvieron relacionadas con el sistema de defensa y la comunicación vía módem.

La familia TMS320 está compuesta por DSPs con multiprocesadores de un solo chip de 16 bits de coma fija, 32 bits de coma flotante y 64 bits. Estos procesadores tienen la

flexibilidad de un controlador de gran velocidad y la capacidad de procesar cadenas numéricas.

Combinando estas dos cualidades, los procesadores TMS320 son una alternativa de bajo coste para la fabricación de procesares mixtos VLSI.

Entre las ventajas más importantes que ofrece Texas Instrument se puede mencionar que existe una amplia información de sus productos a través de su portal en http://www.ti.com, una red de fabricantes que hacen uso de sus DSP y ofrecen sistemas de desarrollo para aplicaciones específicas, además de software y librerías de aplicación que permiten optimizar el código de programa.

Las siguientes características hacen de esta familia una elección ideal para un amplio variedad de aplicaciones de procesado digital de la señal: flexibilidad de programación, flexibilidad de operación, gran rendimiento a alta velocidad, innovadora arquitectura paralela, buena relación coste–efectividad.

Actualmente la familia TMS320 está compuesta por las siguientes generaciones: Generación C6000, generaciones C5000 y generación C2000. Todas estas generaciones se complementan con productos de mezcla de señales como son los convertidores de datos.

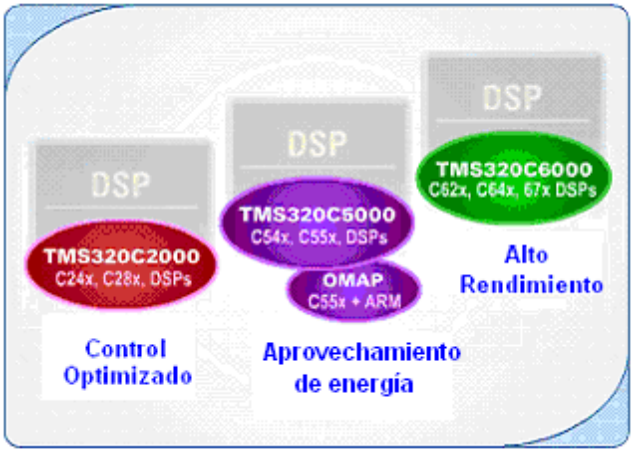


Figura 7-1 Familias de DSPs de Texas Instruments.

Cada generación de integrados TMS320 tiene la misma estructura de CPU combinada con una variedad de memoria incorporada y configuración de periféricos. Se utilizan nuevas combinaciones de memoria incorporada en el encapsulado y opciones de periféricos para crear integrados derivados los cuales satisfacen una amplia gama de necesidades en el mercado mundial de la electrónica.

Además, cuando la memoria y periféricos se integran en un solo procesador, el coste del sistema completo se reduce enormemente y permiten circuitos más reducidos.

Dado que cada generación de DSPs sigue una línea de diseño y está encaminada hacia algún campo de aplicación específico, además cada integrado tiene la capacidad de ajustarse más en profundidad a un proceso determinado debido a sus características especiales.

La nomenclatura por la cual se definen todos los modelos de DSP de la firma Texas Instruments se resume en la figura 7-2.

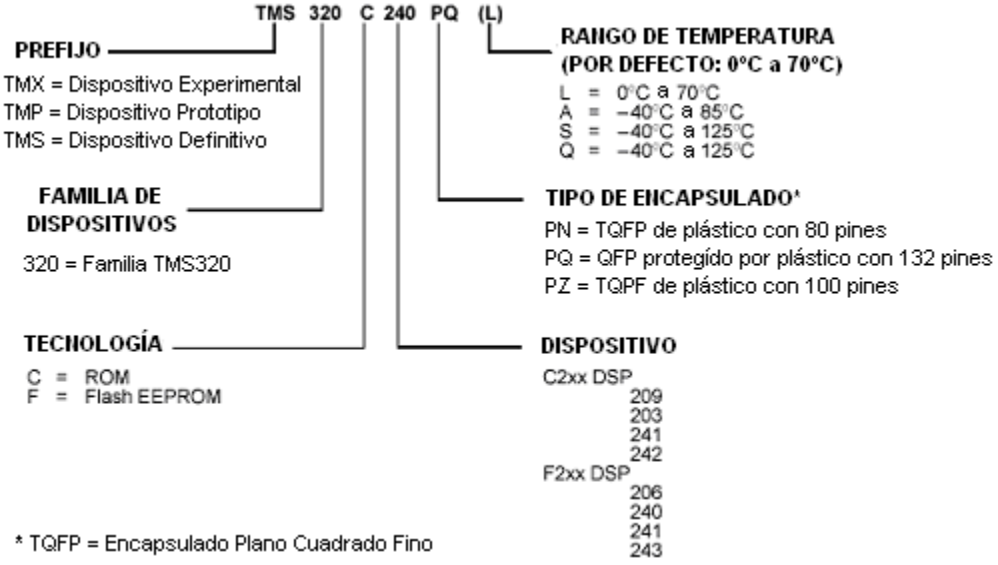


Figura 7-2 Nomenclatura de los DSPs de Texas Instruments.

La generación C6000 marca un hito en nivel de rendimiento y bajo coste. La plataforma de DSPs C6000 ofrece un amplio abanico de dispositivos de los más rápidos de la industria lo cuales funcionan con una frecuencia de reloj de hasta 1 GHz. Esta generación está compuesta por dos series de DSPs de coma fija que son el TMS320C64x y el TMS320C62x y una tercera serie con tecnología de coma flotante que es la TMS320C67x. Estos modelos de DSPs son óptimos para aplicaciones referentes al manejo del ancho de banda en las comunicaciones y desarrollo de aplicaciones de imagen y sonido, ya que tienen una capacidad de procesado de 1200 a 8000 MIPS (millones de instrucciones por segundo) para los modelos de coma fija, y de 600 a 1350 MIPS para los de coma flotante.

Esta serie de DSPs de coma flotante ofrece a los diseñadores de aplicaciones de gran precisión la velocidad, precisión y ahorro de energía que requieren sus proyectos. Estos DSPs dinámicos son la solución ideal para aplicaciones muy demandadas como son la música, adquisición de imágenes en medicina, instrumentación y automóviles.

Las características generales de los dispositivos de la plataforma TMS320C6000 las resumo a continuación:

- La plataforma de dispositivos C6000 tienen un completo conjunto de herramientas de desarrollo como son un eficiente compilador en C y C++, un optimizador para la programación en ensamblador y una interfaz para un depurador basado en Windows.

- Los core CPU de estos dispositivos consisten en dos líneas de registros de propósito general, A y B (ambos para palabras de 32-bit de longitud) y ocho unidades funcionales (dos multiplicadores y seis unidades aritmético lógicas, ALUs). Ejecutan ocho instrucciones de 32-bit por ciclo, diez veces el rendimiento de otros DSPs.

- El empaquetamiento de instrucciones permite ejecutar ocho instrucciones en serie o en paralelo y reduce el tamaño del código, el lanzamiento de instrucciones y el consumo de energía.

- La ejecución condicional de todas las instrucciones reduce el coste en los saltos e incrementa el paralelismo.

- Ejecución de código en unidades funcionales independientes.

- Tienen el compilador en C y C++ más eficiente para DSPs del mercado así como el primer optimizador para ensamblador con el objetivo de conseguir un rápido desarrollo y una mejora en el paralelismo de nuestras aplicaciones.

- Soporta datos para 8/16/32-bit, proporcionando un soporte eficiente de la memoria para una amplia gama de aplicaciones, así como opciones para aritmética con 40-bit que añade una precisión extra para aplicaciones que requieren una alta computacionalidad.

- Aritmética de saturación y normalización.
- El Hardware soporta instrucciones en precisión simple (32-bit) o doble precisión (64-bit).
- Multiplicaciones de 32 x 32 bit con un resultado de 32 o 64-bit.

7.2.1. Procesador TMS320C6713

Los procesadores TMS320C67x están basados en el sistema "VelociTI" desarrollado por Texas Instruments, el cual consiste en una avanzada arquitectura de Instrucciones con Palabras Muy Largas (VLIW). Esta arquitectura altamente paralela y determinista, incrementa la flexibilidad del trabajo del software así como el rendimiento del código mediante el uso de uno de los compiladores de C y optimizadores de ensamblado más eficientes de la industria.

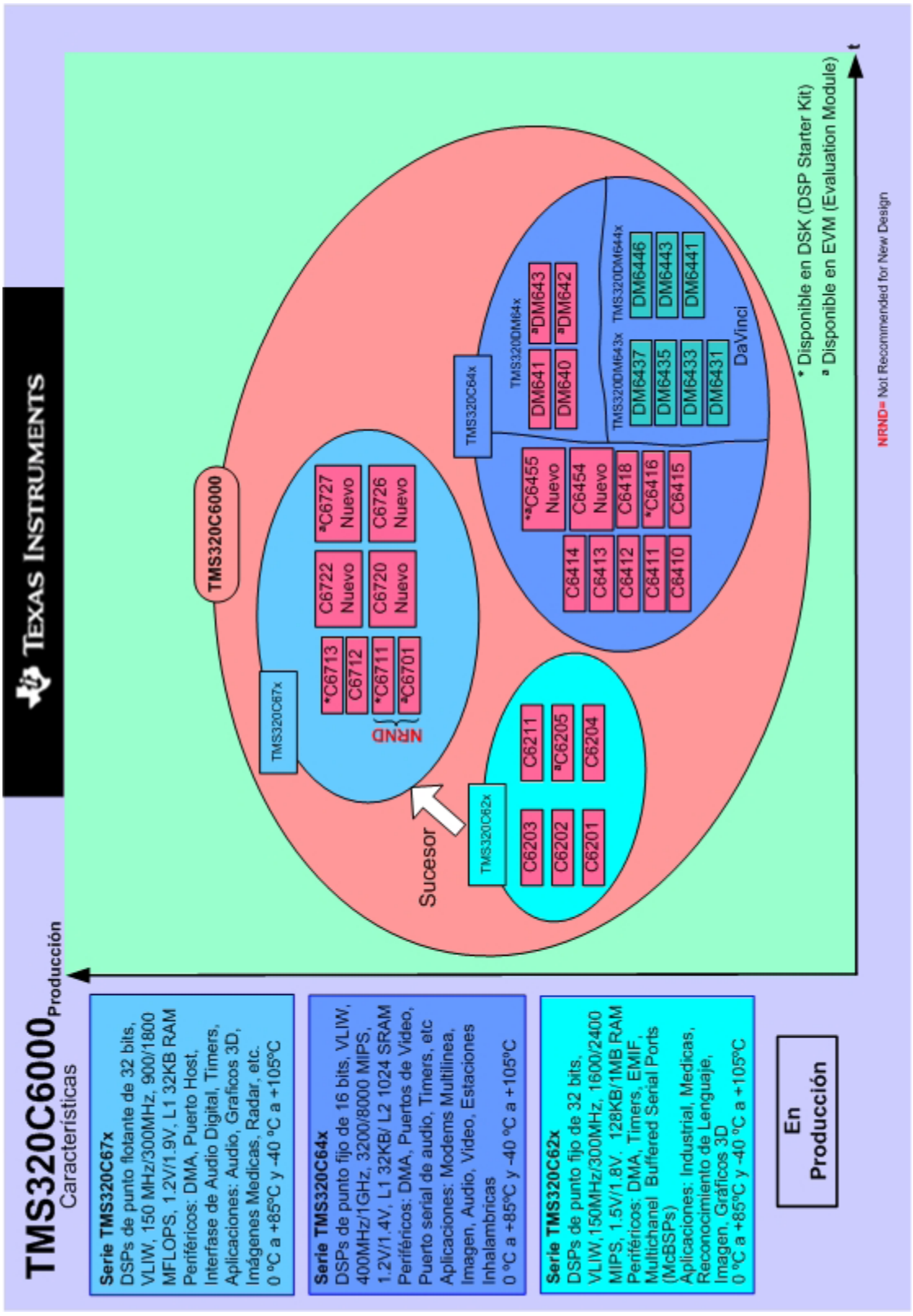


Figura 7-3 Familia de DSP TMS320C6000 del fabricante Texas Instruments.

Las principales características de esta familia las muestro a continuación:

Tabla 7-1 Características de la Serie TMS320C67x.

Modelo	CPU	FREC. (MHz)	RAM	L1/SRAM Interna	ROM	L2/SRAM Interna	EMIF	Memoria Externa Compatible	DMA	HPI	McBSP	McASP	I2C	SPI	Temporizadores	Alimentación del núcleo (V)	Alimentación de IO (V)	Temperatura de Trabajo (°C)	Pin/Encapsulado	Descripción
TMS320C6727-300	1 C67x+	300	256 KB	32 KB	384 KB		1 32-Bit	Asinc. RAM/ROM, SDR SDRAM	dMAX			3	2	2	1 RTI	1.2 V	3.3 V	0 to 90		DSP de coma flotante
TMS320C6727-250	1 C67x+	250	256 KB	32 KB	384 KB		1 32-Bit	Asinc. RAM/ROM, SDR SDRAM	dMAX			3	2	2	1 RTI	1.2 V	3.3 V	-40 to 105, 0 to 90		DSP de coma flotante
TMS320C6726-250	1 C67x+	250	256 KB	32 KB	384 KB		1 16-Bit	Asinc. RAM/ROM, SDR SDRAM	dMAX			3 (McASP 2 DIT solo)	2	2	1 RTI	1.2 V	3.3 V	0 to 90		DSP de coma flotante
TMS320C6726-225	1 C67x+	225	256 KB	32 KB	384 KB		1 16-Bit	Asinc. RAM/ROM, SDR SDRAM	dMAX			3 (McASP 2 DIT solo)	2	2	1 RTI	1.2 V	3.3 V	-40 to 105		DSP de coma flotante
TMS320C6722-250	1 C67x+	250	128 KB	32 KB	384 KB		1 16-Bit	Asinc. RAM/ROM, SDR SDRAM	dMAX			2	2	2	1 RTI	1.2 V	3.3 V	0 to 90		DSP de coma flotante
TMS320C6722-225	1 C67x+	225	128 KB	32 KB	384 KB		1 16-Bit	Asinc. RAM/ROM, SDR SDRAM	dMAX			2	2	2	1 RTI	1.2 V	3.3 V	-40 to 105		DSP de coma flotante
TMS320C6722-200	1 C67x+	200	128 KB	32 KB	384 KB		1 16-Bit	Asinc. RAM/ROM, SDR SDRAM	dMAX			2	2	2	1 RTI	1.2 V	3.3 V	0 to 90		DSP de coma flotante
TMS320C6713-300	1 C67x	300		8 KB		64 KB	1 32-Bit	Asinc. SRAM, SBSRAM, SDRAM	16-Ch EDMA	1 16-Bit	2	2	2		2 32-Bit GP	1.4 V	3.3 V	0 to 90	272BGA	DSP de coma flotante
TMS320C6713-225	1 C67x	225		8 KB		64 KB	1 32-Bit	Asinc. SRAM, SBSRAM, SDRAM	16-Ch EDMA	1 16-Bit	2	2	2		2 32-Bit GP	1.26 V	3.3 V	0 to 90	272BGA	DSP de coma flotante
TMS320C6713-200	1 C67x	200		8 KB		64 KB	1 16-Bit, 1 32-Bit	Asinc. SRAM, SBSRAM, SDRAM	16-Ch EDMA	1 16-Bit	2	2	2		2 32-Bit GP	1.2 V/1.26 V	3.3 V	-40 to 105, 0 to 90	208HLQFP, 272BGA	DSP de coma flotante

Modelo	CPU	FREC. (MHz)	RAM	L1/SRAM Interna	ROM	L2/SRAM Interna	EMIF	Memoria Externa Compatible	DMA	HPI	McBSP	McASP	I2C	SPI	Temporizadores	Alimentación del núcleo (V)	Alimentación de IO (V)	Temperatura de Trabajo (°C)	Pin/Encapsulado	Descripción
TMS320C6713-167	1 C67x	167		8 KB		64 KB	1 16-Bit	Asinc. SRAM, SBSRAM, SDRAM	16-Ch EDMA	1 16-Bit	2	2	2		2 32-Bit GP	1.2 V	3.3 V	-40 to 105	208HLQFP	DSP de coma flotante
TMS320C6712-150	1 C67x	150		8 KB		64 KB	1 16-Bit	Asinc. SRAM, SBSRAM, SDRAM	16-Ch EDMA		2				2 32-Bit GP	1.26 V	3.3 V	0 to 90		DSP de coma flotante
TMS320C6712-100	1 C67x	100		8 KB		64 KB	1 16-Bit	Asinc. SRAM, SBSRAM, SDRAM	16-Ch EDMA		2				2 32-Bit GP	1.8 V	3.3 V	0 to 90		DSP de coma flotante
TMS320C6711-250	1 C67x	250		8 KB		64 KB	1 32-Bit	Asinc. SRAM, SBSRAM, SDRAM	16-Ch EDMA	1 16-Bit	2				2 32-Bit GP	1.4 V	3.3 V	0 to 90	272BGA	DSP de coma flotante
TMS320C6711-200	1 C67x	200		8 KB		64 KB	1 32-Bit	Asinc. SRAM, SBSRAM, SDRAM	16-Ch EDMA	1 16-Bit	2				2 32-Bit GP	1.26 V	3.3 V	0 to 90	272BGA	DSP de coma flotante
TMS320C6711-167	1 C67x	167		8 KB		64 KB	1 32-Bit	Asinc. SRAM, SBSRAM, SDRAM	16-Ch EDMA	1 16-Bit	2				2 32-Bit GP	1.26 V	3.3 V	-40 to 105, 0 to 90	272BGA	DSP de coma flotante
TMS320C6701-167	1 C67x	167	128 KB				1 32-Bit	Asinc. SRAM, SBSRAM, SDRAM	4-Ch DMA	1 16-Bit	2				2 32-Bit GP	1.9 V	3.3 V	-40 to 105, 0 to 90		DSP de coma flotante
TMS320C6701-150	1 C67x	150	128 KB				1 32-Bit	Asinc. SRAM, SBSRAM, SDRAM	4-Ch DMA	1 16-Bit	2				2 32-Bit GP	1.8 V	3.3 V	-40 to 105, 0 to 90	352FC/CSP	DSP de coma flotante

La arquitectura VelociTIE es una arquitectura avanzada de VLIW de alto rendimiento cuyas características más importantes son: el empaquetado de instrucciones con lo que se reduce el tamaño del código; todas las instrucciones pueden operar de forma condicional lo cual aporta gran flexibilidad al código y tamaño variable de los datos en las instrucciones.

Para mi estudio he utilizado el DSP TMS320C6713 cuyo diagrama de bloques es el de la figura 7-4 [105]:

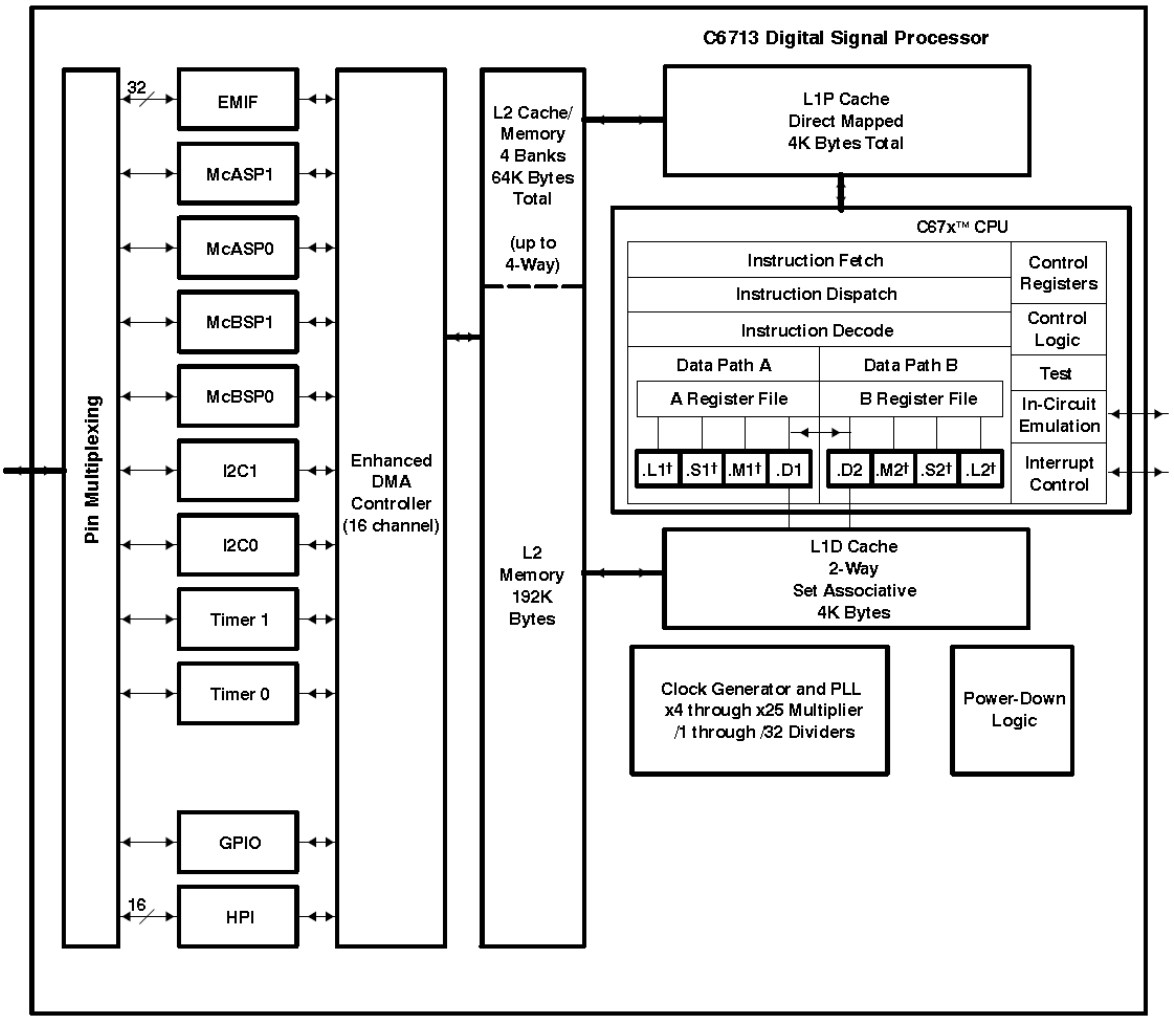


Figura 7-4 Bloques funcionales y diagrama de la CPU (*dsp core*) del DSP TMS320C6713 de Texas Instruments.

El DSP TMS320C6713 de Texas Instrument. Es un procesador de punto flotante de 32 bits. Las unidades "*instruction fetch*", "*instruction dispatch*" y la "*instruction decode*" pueden enviar hasta 8 instrucciones de 32 bits a las unidades funcionales en cada ciclo de reloj. Como puede verse en la figura 7-4, el procesamiento de las instrucciones puede realizarse por dos vías distintas (A o B). Cada vía cuenta con 4 unidades funcionales (.L, .S, .M, y .D) y 16 registros de propósito general de 32 bits. Los core CPU de estos dispositivos consisten en dos líneas de registros de propósito general, A y B (ambos para

palabras de 32 bit de longitud) y ocho unidades funcionales (dos multiplicadores y seis unidades aritmético lógicas, ALUs). Ejecutan ocho instrucciones de 32 bits por ciclo. Puede realizar multiplicaciones de 32 x 32 bit con un resultado de 32 o 64 bit. La CPU contiene: unidad de búsqueda (*fetch*) de programa, unidad de envío (*dispatch*) de instrucción, unidad de decodificación de instrucción, dos rutas de datos cada una con cuatro unidades funcionales. Estas unidades de *fetch*, *dispatch* y *decode* son capaces de distribuir hasta 8 instrucciones de 32 bits a las unidades funcionales correspondientes en cada ciclo de reloj de la CPU. El procesamiento de las instrucciones ocurre en cada una de las dos rutas de datos (A y B), cada una de las cuales contiene cuatro unidades funcionales (.L, .S, .M y .D) y 16 registros de propósito general de 32 bits.

La etapa de *fetch* tiene cuatro fases para todas las instrucciones y la etapa de *decode* tiene dos. La etapa de *execute* requiere un número de fases que varía dependiendo del tipo de instrucción. Las fases del *fetch* son las siguientes: PG (generación de la dirección de programa), PS (envío de la dirección de programa), PW (espera del acceso a programa) y PR (recibo del *fetch packet* del programa). Las fases de *decode* son DP (envío de la instrucción o *dispatch*) y DC (*decode* de la instrucción). La ejecución en punto flotante del *pipeline* está dividida en diez fases (E1-E10), en punto fijo estará divida en cinco fases. Diferentes tipos de instrucciones requieren un diferente número de fases para completar su ejecución.

La CPU contiene además 32 registros de 32 bits, registros de control, lógica de control y lógica para la simulación, test y control de las interrupciones.

7.3. Características del ARM (Advanced RISC Machine)

La arquitectura ARM es una alternativa muy importante en el diseño de procesadores, por que permite usar un número menor de instrucciones, alberga menos espacio de memoria y debido a su procesador de 32 bits puede desempeñar muchas mas aplicaciones que otras arquitecturas. Gracias a su diseño sencillo, el ARM tiene relativamente pocos componentes en el chip, por lo que no alcanza altas temperaturas y tiene bajos requerimientos de energía. Estas características lo han hecho candidato perfecto para el mercado de aplicaciones empotradas.

La arquitectura ARM es líder en la industria de procesadores de 32 bits y es la más empleada en la mayoría de los dispositivos electrónicos que existen en la actualidad . Con

más de 600 socios que adoptan esta arquitectura para desarrollar herramientas, software y procesadores que son utilizados en automóviles, Smartphones, PDAs, Notebooks, Laptops, HDTV, impresoras, reproductores multimedia, equipos médicos, nanosatelites, etc., han hecho imprescindible el considerar a la arquitectura ARM como base en el diseño de nuevos productos.

Es importante tener en cuenta que esta arquitectura posee características relevantes como un Conjunto de Instrucciones Reducido (RISC) de tamaño fijo con un formato de 3 direcciones, lo cual facilita la labor del decodificador de instrucciones haciéndolo más simple. Utilizan la técnica del *pipeline* permitiendo así optimizar los recursos de hardware y también el rendimiento del procesador al comenzar a procesar una instrucción antes de que se haya finalizado de procesar la actual. Tienen control sobre la Unidad Aritmética Lógica (ALU) y disponen de un desplazador (*shifter*) que es utilizado para ejecutar operaciones complejas en una sola instrucción. Además todas las instrucciones se ejecutan en un ciclo de reloj, posee modos de direccionamiento simples y utiliza arquitectura de almacenamiento y carga (*load/store*), donde las instrucciones que procesan datos lo hacen solamente a través de registros y separadas de las instrucciones que acceden a memoria. Estas características permiten implementaciones de tamaño reducido, de muy bajo consumo de energía y de alto rendimiento.

La arquitectura ARM fue creada por Acorn Computer Group, como el primer procesador RISC con gran impacto comercial en el mundo. La filosofía RISC (*Reduced Instruction Set Computer*) busca realizar un proceso de la manera menos complicado posible.

El ARM es un procesador RISC de 32 bits usado en un amplio número de aplicaciones, son los procesadores más populares en el mundo utilizados en sistemas empotrados. Hoy en día, cerca del 75% de los procesadores de 32 bits poseen este chip en su núcleo. La arquitectura ARM es, por decirlo de alguna manera, la arquitectura más reconocida en el mundo de los sistemas empotrados, gracias a su buena relación de potencia/consumo.

Las diferentes familias de arquitectura ARM están estandarizadas por un núcleo procesador y periféricos estándares que ayudan a manejar la complejidad y compatibilidad de diferentes sistemas. Según se han ido desarrollando las diferentes familias de la

arquitectura ARM como son ARM7, ARM9, ARM10, ARM11, han ido incrementado los adelantos tecnológicos, brindado un incontable número de herramientas para desarrollar proyectos con esta arquitectura. La investigación de la arquitectura ARM concederá bases teóricas y herramientas para el desarrollo de proyectos, lo que ayudará a que los ingenieros electrónicos realicen sus proyectos de forma eficiente y con tecnologías actuales.

La mayoría de las compañías electrónicas internacionales han tomado la arquitectura ARM como el principal instrumento para el adelanto tecnológico, debido a que ofrece un extenso número de productos para el desarrollo de proyectos como; microprocesadores RISC de 16/32 bits, procesador de datos, procesador 3D, librerías digitales, memorias integradas, periféricos, software, herramientas de desarrollo; muchas de las cuales son de libre acceso. Debido a esto es necesario hacer un estudio de la arquitectura ARM, sus diferentes familias, fabricantes y herramientas de desarrollo para sentar bases teóricas con la finalidad de conocer los beneficios que brinda y poder desarrollar diferentes tipos de aplicaciones de acuerdo con los avances electrónicos que se den en el mundo.

7.3.1. Historia del ARM

El primer circuito integrado ARM fue desarrollado para los ordenadores de Acorn, uno de los pioneros en el desarrollo de microcomputadores. En esos tiempos Arcon era una de las principales marcas de ordenadores personales británicos. El éxito inicial de Arcon se estableció cuando la Corporación de Radiodifusión Británica (BBC) creó un nuevo modelo de ordenador personal que fue comercializado como el microcomputador BBC. El lanzamiento de este micro BBC en 1982 le hizo pionero en la etapa inicial de los ordenadores personales principalmente en Gran Bretaña, y el nombre BBC le dio al diseño Acorn más importancia y credibilidad frente a otros competidores. El micro BBC fue la base para el procesador 6502 de 8 bits de Rockwell, el mismo chip que impulsó el Apple II. Los modelos iniciales de Arcon ofrecían gráficos en color y una memoria RAM de 32 Kbytes.

Los primeros desarrollos del chip ARM comenzaron en 1983, para continuar en 1985 con Steve Furber, Roger Wilson y Robert Heaston que formaron el grupo de trabajo para desarrollar un dispositivo que reuniera las características del procesador 6502 pero en un entorno RISC de 32 bits e implementarlo en un dispositivo pequeño que fuese fácil de diseñar, probar y fabricarlo con un coste mínimo. La decisión más importante fue utilizar

instrucciones de tamaño constante, un modelo para cargar los datos además de utilizar un conjunto de instrucciones reducido.

El primer conjunto de instrucciones ARM fue escrito en BASIC, por lo que los siguientes modelos ARM también fueron realizados en el mismo lenguaje fuente. El diseño físico del chip fue realizado usando tecnología VLSI que era utilizado normalmente como herramienta de programación, para realizar pruebas de los circuitos integrados se diseño un simulador llamado *even–driven* el cual fue hecho en BASIC, así como también fueron diseñados y probados el controlador de video (VIDC), el controlador de memoria (MEMC) y el controlador de I/O (IOC). Posteriormente se diseñaría un nuevo simulador contruido en Modula–2 y posteriormente en C que fue conocido como ASIM, el cual es usado actualmente para el diseño y prueba de dispositivos por las compañías ACORN y ARM Ltd.

El primer procesador comercial RISC y primer procesador ARM fue el ARM1, el cual se fabricó por primera vez en Abril de 1985 con tecnología VLSI , éste mejoró las metas de diseño planteadas ya que usó menos de 25.000 transistores y se diseñaron usando 3 microprocesadores. Posteriormente se encontró que en la arquitectura ARM1 existían algunas áreas donde el conjunto de instrucciones podía mejorarse para aumentar el desarrollo de los sistemas basados en esta arquitectura. La investigación de los laboratorios Acorn albergó el primer ejemplar de una nueva familia de procesadores RISC. Sin embargo los procesadores ARM fueron creados con la intención de potenciar la siguiente generación de computadores personales Acorn.

En 1987, una casa de computadores, *Archimedes*, lanzó el primer chip comercial usando la arquitectura ARM, ofreciendo una versión de 8 MHz de la ARM2, con un controlador de entradas/salidas y un sistema operativo muy simple. Los *Archimedes* no tuvieron una buena acogida en su lanzamiento; porque aparecieron los ordenadores personales con los estándares de los PC IBM mientras Acorn introducía un computador con un nuevo procesador, un nuevo sistema operativo y no tenía un software base disponible para el usuario. El procesador ARM2 podía trabajar en cuatro modos: USR (modo usuario), IRQ (modo de interrupción), FIQ (modo de interrupción rápida) y SVC (modo supervisor). Del modo de usuario no se podía cambiar al modo del procesador existiendo un hardware de seguridad y la memoria física era solo accesible por medio de

un código propio de la compañía. Los dos últimos bits del registro de estado indican el modo de ooperación del procesador. (00 – USR, 01 – IRQ, 10 FIQ y 11 - SVC).

El ARM3 está constituido por una macrocelda basada en el núcleo de la ARM2 y además se incluyó un interfaz especializado de coprocesador, el registro de etado no se modificó y no hay nuevos modos de procesador. En este chip se agregó una memoria caché con lo que se consiguió una velocidad de reloj mucha más rápida. También se le hicieron ajustes en la interfaz del coprocesador en el chip disponiendo de 15 definiciones del coprocesador para disponer de un control de la memoria caché y una identificación del chip. Además se añadió una nueva instrucción, la instrucción SWP, utilizada para intercambiar datos en la memoria.

El interés en las familias ARM fue creciendo, los diseñadores se veían más interesados en RISC y los diseños de ARM eran vistos como una necesidad para obtener un alto desarrollo, baja potencia de consumo y procesadores RISC de bajo costo. A partir de un acuerdo entre las compañías Arcon, VLSI Technology Inc. y Apple nació la compañía Arcon RISC Machine se denominó Advance RISC Machine Ltd. ARM Ltd. fue fundada con la clara misión de seguir con el desarrollo de procesadores ARM y facilitar su uso para el desarrollo de sistemas. En este cambio se saltaron las versiones 4 y 5, apareciendo el ARM6. Este es el primer chip el cual estuvo disponible comercialmente de todos los ARMs, este tenía una capacidad de direccionamiento de 32 bits, además el procesador tenia 31 registros con seis nuevos modos de procesador.

Hoy en día, ARM ha centrado su arquitectura en el mercado de sistemas empotrados, llegando a tener un 90% del mercado de procesadores de 32 bits de tecnología RISC. Es destacable su uso en electrónica de consumo, como móviles, tabletas, videoconsolas, y periféricos como routers o controladoras para equipos informáticos, e incluso se está empezando a plantear el uso de estos procesadores en entornos de supercomputación y servidores a gran escala. El modo de funcionamiento de ARM está licenciado (como ocurre con los IP cores), constando de una descripción hardware del núcleo del procesador, un entorno completo de desarrollo (compilador, debugger y SDK) y la posibilidad de comercializar productos con el procesador ARM integrado.

7.3.2. Familias de la arquitectura ARM

ARM ha especializado la arquitectura para los distintos segmentos del mercado. En la actualidad existen las siguientes series de procesadores ARM:

- **Series Cortex-A:** Procesadores de alto rendimiento, diseñados para sistemas operativos abiertos. Son multiprocesadores de alta frecuencia, con varias unidades de coma flotante y SIMD.

- **Series Cortex-R:** Procesadores exclusivos para aplicaciones con funcionamiento en tiempo real, con bajo consumo y alto rendimiento.

- **Series Cortex-M:** Procesadores dedicados al uso como microcontroladores, tienen un tamaño reducido, y ofrecen el menor consumo posible.

- **Series Classic:** Son las arquitecturas ARM7, ARM9, ARM10E y ARM11. Ofrecen un precio reducido.

- **Series especiales:** Son las arquitecturas *SecurCore*, que contienen módulos específicos de seguridad, y las arquitecturas para FPGAs, que contienen diseños específicos para su implementación en FPGAs como IP *cores*. El primero de la serie de procesadores específicos para FPGAs es el ARM Cortex-M1, y puede implementarse en FPGAs de Altera, Actel y Xilinx.

En la figura 7-5 se observa una clasificación de los núcleos ARM en función del rendimiento/funcionalidad y capacidad. De igual forma se muestran con colores el tipo de aplicaciones en los que son utilizados estos núcleos.[106]

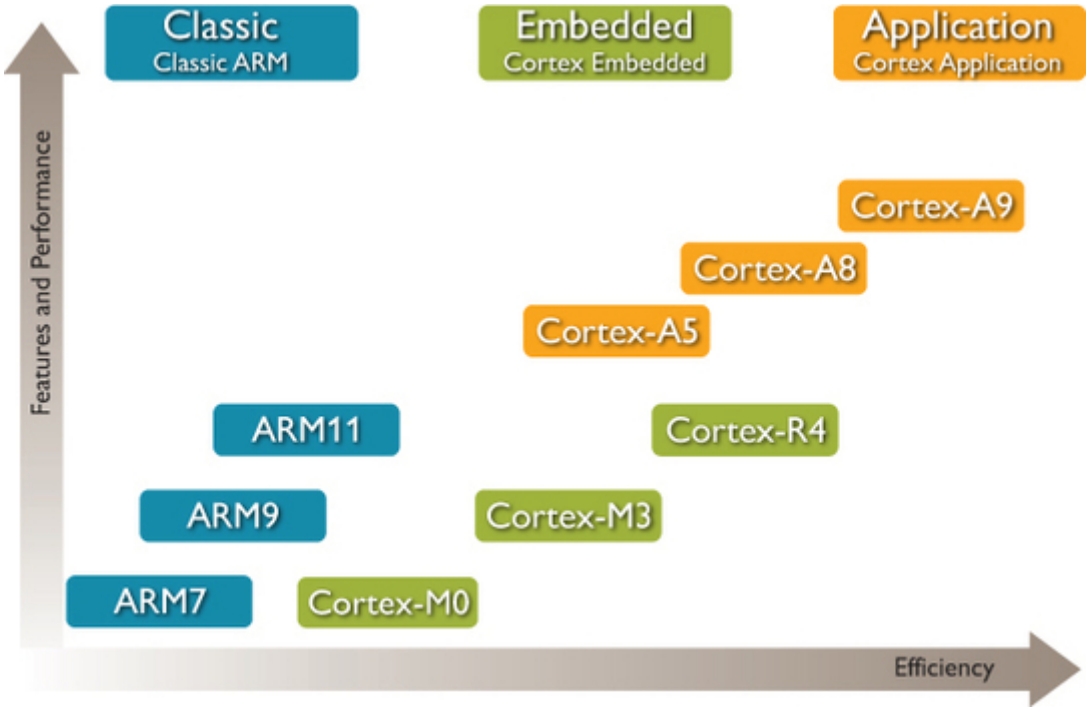


Figura 7-5 Clasificación de los núcleos ARM.

En la figura 7-6 se observa una clasificación que muestra las versiones de la arquitectura ARM y algunas de las tecnologías complementarias utilizadas por los núcleos. Tecnologías como NVIC (*Nested Vectored Interrupt Controller*), WIC (*Wakeup Interrupt Controller*), Thumb2 son determinantes en procesadores para sistemas empotrados que necesitan comportamiento estable, un muy bajo consumo de energía y un número pequeño de pines (menos de 144) mientras se mantiene una alta eficiencia de procesado y una amplia gama de periféricos (interfaces serie de alta velocidad, temporizadores, controlador de motores, etc).

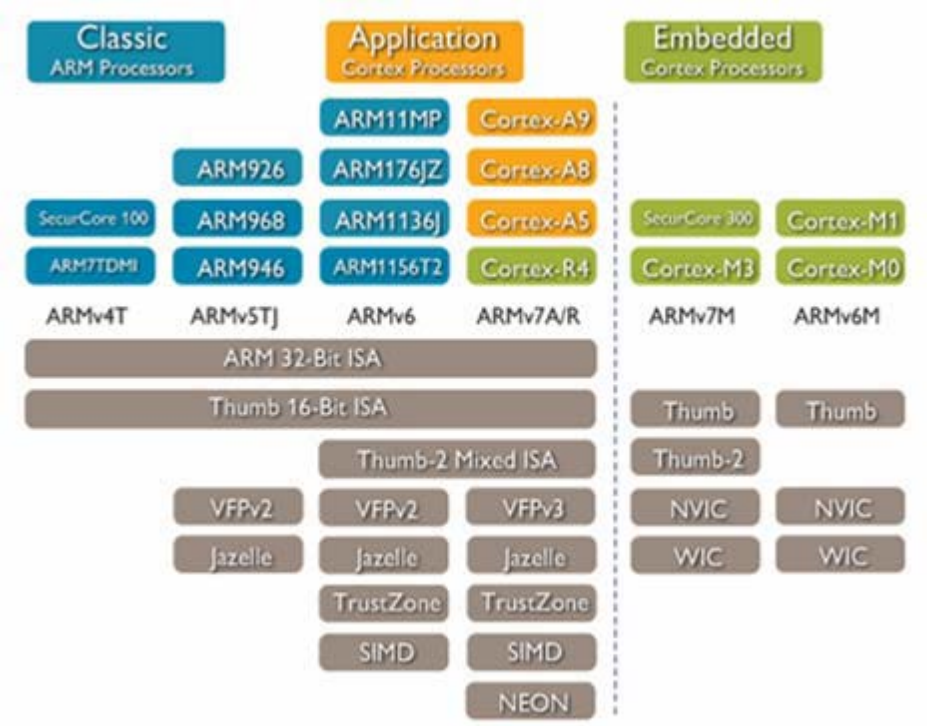


Figura 7-6 Versiones y Tecnologías complementarias en los núcleos ARM.

Los procesadores que he utilizado en este capítulo de la tesis para la comparación de la operación de convolución son el clásico ARM7 y el ARM Cortex M3. A continuación realizo una descripción de sus características principales.

7.3.2.1. Arquitectura de la familia ARM7

La familia ARM7, introducida en 1994, consta de un microprocesador RISC de 32 bits, trabaja sobre los 130 MIPS, e incorpora el conjunto de instrucciones de 16 bits de la tecnología *Thumbs* consiguiendo un rendimiento de 32 bits con un sistema de 8 o 16 bits.

La familia ARM7 esta compuesta por los siguientes modelos de núcleos procesadores [107]: ARM7TDMI, ARM7TDMI-S, ARM7EJ-S y el ARM720T, cada uno de los cuales tienen que ser desarrollados en base a los diferentes requerimientos de mercado como son: versión sintetizable de la versión del procesador ARM7TDMI, núcleo sintetizable con tecnología DSP y *Jazelle* encargados de mejorar el rendimiento en Java, y núcleo con unidad de manejo de memoria (MMU) que sirve de apoyo para trabajar con sistemas operativos tales como Windows CE, Palm OS, Symbian OS y Linux. En mis datos experimentales he utilizado el ARM7TDMI-S.

Como aplicaciones principales de los ARM7 cabe destacar su utilización en dispositivos de audio como: MP3, WMA, AAC players, en los microteléfonos inalámbricos, *beepers*, etc.

Sus Características principales son las siguientes:

- Posee una arquitectura RISC de 32 bits
- Tiene un rendimiento sobre los 130 MIPs con un procesador típico de 0.13µm.
- Diseño pequeño y muy bajo consumo de potencia
- Programación de alto nivel, comparable con los microcontroladores de 16 bits
- Sistema operativo compatible como Windows CE, Palm OS, Symbian OS y Linux.
- Posee un amplio número de herramientas de desarrollo disponibles.
- Simulación acorde para los principales entornos de desarrollo EDA.
- Excelente depuración de errores para los diseños SoC.
- Migración y soporte a través de nuevos procesos tecnológicos.
- Su código es compatible con los procesadores ARM9, ARM9E, ARM10.

La constitución interna del ARM7 se indica en la figura 7-7:

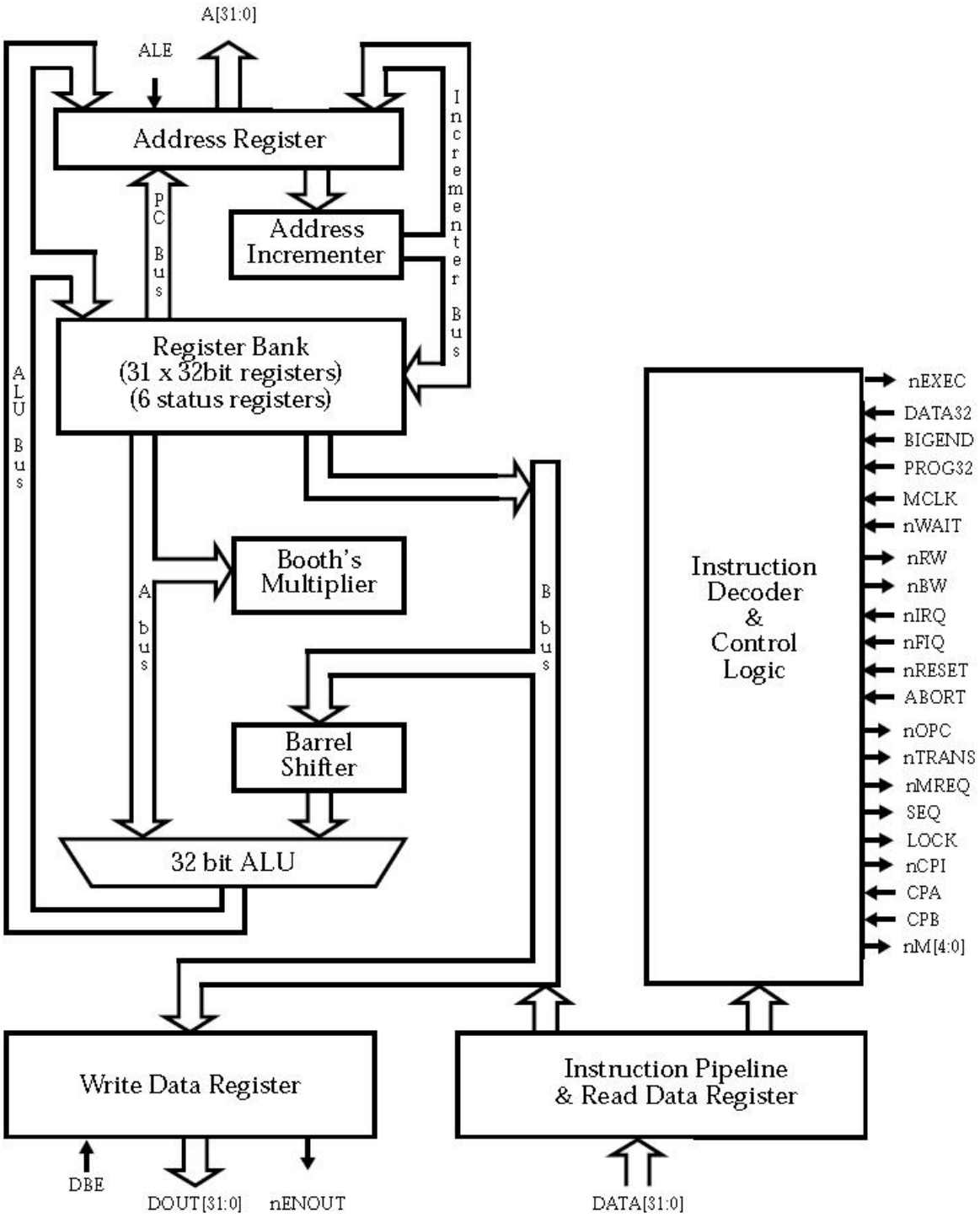


Figura 7-7 Diagrama de bloques del ARM7

Está formado por dos bloques principales el *datapath* y el *decoder*. Posee un banco de registros desde r0 a r15, dos puertos de lectura que están conectados al bus de datos A o al B, un puerto de escritura, puertos adicionales de lectura o escritura para el contador de programa r15, barrel shifter el cual permite rotar o mover el segundo operando en cualquier número de bits, unidad aritmética lógica, y consta de registros de direcciones de operandos.

El ARM7TDMI tiene una arquitectura Von Neumann, con un único bus de datos de 32 bits de longitud que transporta tanto instrucciones como datos. Sólo las instrucciones load, store y swap pueden acceder a los datos desde memoria. Estos datos pueden tener una longitud de 8, 16 o 32 bits. La interfaz está diseñada para permitir mejoras potenciales mientras se minimiza el uso de la memoria. Además, permite la integración en dispositivos con coprocesadores.

Los procesadores ARM tienen un hardware adicional llamado *EmbeddedICELogic* que, haciendo las veces de depurador ofrece una serie de herramientas software para poner a punto código que se ejecute sobre el procesador.

El funcionamiento de este procesador se podría resumir de la forma siguiente: El registro de datos retiene los datos que van a ser leídos de la memoria o escritos en la memoria, el decodificador de instrucciones decodifica el código de instrucciones de la máquina para producir señales de control al *datapath*. En un ciclo simple de proceso de datos, se puede notar que los valores de los datos son leídos en los buses A y B y los resultados de la unidad aritmética lógica son escritos nuevamente dentro del banco de registros. El valor del contador de programa es incrementado y copiado nuevamente al registro r15, esto permite obtener un adelanto en el tiempo para que se pueda ejecutar otra instrucción.

Por otro lado el *pipeline* es la parte fundamental en esta arquitectura debido a que incrementa la velocidad de ejecución de las instrucciones, es decir ejecuta más instrucciones en un sólo ciclo de reloj. La arquitectura ARM utiliza 3 etapas para realizar el pipeline. En la etapa de *fetch* extrae el código de instrucción desde la memoria dentro de la instrucción pipeline. En la etapa *decode* decodifica la instrucción para obtener las señales de control para el datapath quedando listo para realizar el último paso que consiste en la etapa de ejecución (*execute*) donde se trabaja con instrucciones propias del datapath, se produce la lectura de los registros, la variación de estos y los resultados generados por el ALU son escritos nuevamente en el registro. Los resultados de cada etapa del *pipeline* son guardados en los registros. La ventaja del pipeline es que el periodo de reloj es mucho más pequeño que si trabajara sin *pipeline*.

En cualquier instante de tiempo, tres instrucciones diferentes pueden ocupar cada una de las tres etapas del pipeline, esto puede tomar 3 ciclos para completar una instrucción de

ciclo simple, es decir, tiene tres ciclos latentes. Una vez que el pipeline es realizado, el procesador completa una instrucción de ciclo simple en cada ciclo de reloj. Sin embargo el rendimiento es el de una instrucción por ciclo. Una de las características importantes del *pipeline* es el de eliminar los saltos a subrutinas logrando así obtener un mejor flujo del código.

El core ARM7TDMI tiene una arquitectura ARMv4T (Von Neumann), un *pipeline* de tres estados, la velocidad del *core* es de 150 MHz, y señalar que no dispone de memoria caché.

El ARM7TDMI tiene dos repertorios de instrucciones diferentes: repertorio ARM de 32 bits y repertorio Thumb de 16 bits. Thumb implementa un repertorio de instrucciones de 16 bits en una arquitectura de 32 bits para mejorar el rendimiento de una arquitectura de 16 bits y obtener una mayor densidad de código que en una arquitectura de 32 bits. Las instrucciones Thumb son transformadas, de manera transparente para el programador, en instrucciones de 32 bits en tiempo de ejecución sin pérdida de rendimiento. Por tanto, Thumb tiene todas las ventajas de un núcleo de 32 bits, haciendo que el núcleo del ARM7TDMI sea ideal para aplicaciones de sistemas empotrados que tienen una gran restricción de ancho de banda en memoria.

La disponibilidad de un repertorio de 16 bits y otro de 32 permite a los diseñadores tener flexibilidad entre el tamaño del código y el rendimiento a nivel de subrutinas dependiendo de las necesidades de su aplicación.

7.3.2.2. Arquitectura del Cortex-M3

Los dispositivos *System-On-Chip* basados en procesadores embebidos ARM abarcan un amplio segmento del mercado, la familia Cortex maneja tres distintos perfiles dentro de la arquitectura ARMv7: el perfil A destinado a la ejecución de sistemas operativos complejos; el perfil R para sistemas que requieren tolerancia a fallos; y el perfil M optimizado para sistemas de bajo coste y en aplicaciones con microcontroladores. [106]

El procesador ARM Cortex–M3 es un estándar industrial para aplicaciones empotradas que requieren características de tiempo real (rápida respuesta frente a eventos), alto rendimiento y bajo coste. Ha sido específicamente diseñado para permitir a los socios ARM desarrollar plataformas de alto rendimiento con bajo consumo de energía enfocadas

hacia un amplio rango de dispositivos tales como microcontroladores, sistemas de control industrial, redes inalámbricas y sensores. Tiene un pequeño número de puertas y una baja latencia en el procesamiento de interrupciones.

Este procesador incorpora tecnologías complementarias como NVIC y WIC. El NVIC (*Nested Vector Interrupt Controller*) facilita la baja latencia en el manejo de las interrupciones, controla el manejo de energía e implementa un sistema de registros de control. Soporta hasta 240 interrupciones dinámicamente repriorizables cada una con 256 niveles de prioridad. El NVIC mantiene el conocimiento de la pila de interrupciones (*nested*) para poder ponerlas en cola de espera mientras se terminan de procesar las anteriores. Debido a su implementación no es posible utilizarla en modos de "sueño profundo" (*very-deep-sleep*). Por este motivo se utiliza la tecnología WIC (*Wake–up Interrupt Controller*) para estos modos, permitiendo al procesador despertar y atender a las interrupciones que se encuentren en cola. Debido a su muy pequeña complejidad para ajustarse a ste tipo de modos de operación, no puede manejar prioridades en las interrupciones.

La unidad de manejo de memoria (MMU) que disponen otros procesadores (como el ARM Cortex–A8) es sustituída en este procesador por una MPU (*Memory Protection Unit*) que es utilizada para proteger regiones de memoria, reforzar los privilegios de acceso y separar los procesos. Lo que realiza en la práctica es prevenir que las aplicaciones de usuario puedan acceder a las regiones de memoria del sistema operativo en tiempo real que esté cargado en el procesador.

Por otro lado emplea un conjunto de instrucciones ARM *Thumb*–2 para obtener una densidad de código global comparable con las instrucciones *Thumb* y un rendimiento comparable con las instruciones ARM.

Para obtener un mayor rendimiento el procesador puede trabajar en modo *hard* o trabajar inteligentemente. Aumentar la frecuencia del reloj puede mejorar el rendimiento, pero este hecho va acompañado de un incremento en el consumo de energía y en una complejidad del diseño. Por otro lado en diseños más simples se consigue una mayor eficiencia computacional con frecuencias de reloj más bajas, situación que explotan los Cortex–M3 que disponen de frecuencias de hasta 100 MHz.

El core Cortex–M3 emplea la arquitectura ARMv7–M, más reciente que los tradicionales *cores* ARM7 y ARM9, que empleaban ARMv4 y ARMv5, respectivamente. Funcionalmente guardan muchas similitudes, y comparten gran cantidad de instrucciones, aunque evidentemente ARMv7–M ha sido diseñada más para el ambiente de microcontroladores, y por consiguiente difiere de las anteriores en áreas como manejo de interrupciones y demás excepciones, niveles de ejecución, y manejo de memoria. Este procesador Cortex–M3 incorpora un *pipeline* de 3 etapas y usa una arquitectura *Harvard*, con un bus de instrucciones y otro de datos separados, además de un tercer bus para periféricos. También incorpora una unidad de predicción de saltos. La característica más importante que nos interesa para nuestro estudio es que su ALU dispone de un multiplicador hardware en un solo ciclo y de un divisor hardware, a diferencia del ARM7 clásico.

En la figura 7-8 se muestra una diagrama simplificado de la arquitectura de este procesador [108].

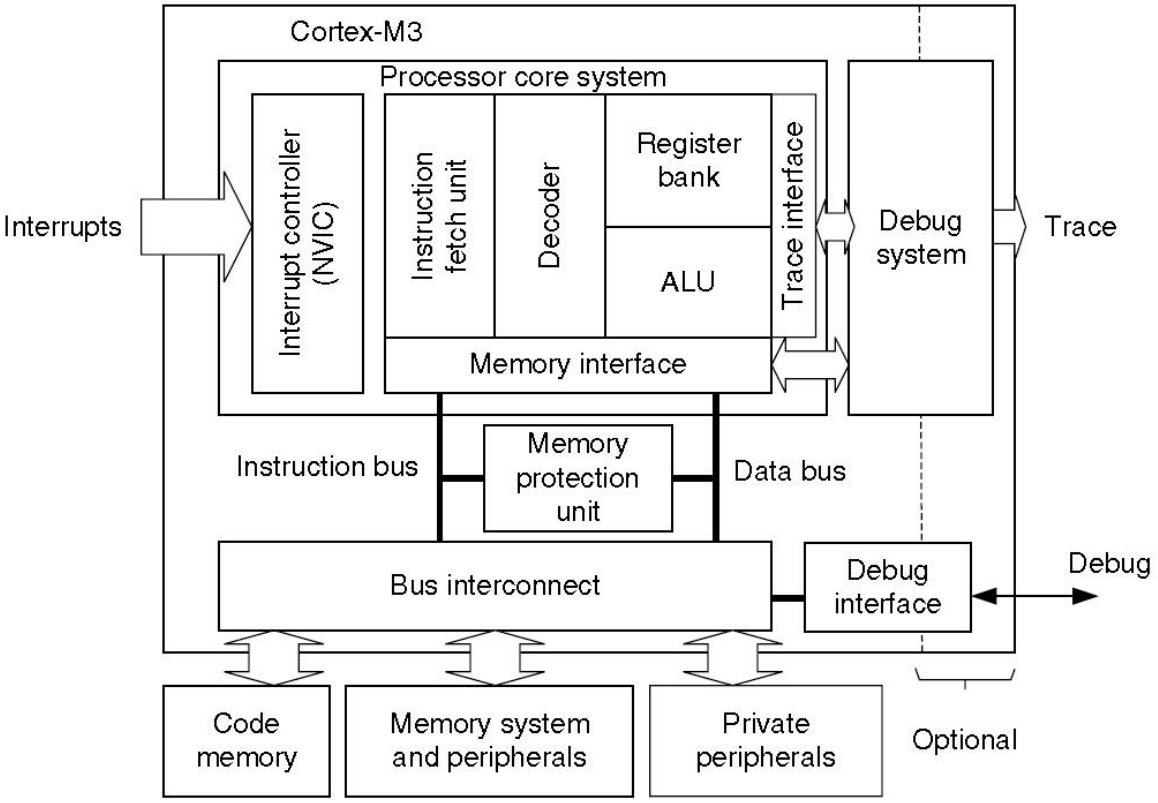


Figura 7-8 Diagrama simplificado del ARM Cortex-M3.

7.4. Resultados obtenidos

En este apartado mostramos los resultados experimentales obtenidos para la operación de la convolución en los cinco dispositivos en estudio.

La utilidad de la operación de convolución es conocer la salida de cualquier sistema lineal e invariante en el tiempo a partir de conocer la respuesta de dicho sistema a la función impulso. La convolución de dos señales viene descrita por la siguiente ecuación:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

El algoritmo utilizado es el propuesto en [5] desde el punto de vista de la salida.

```
For i=0 to M+N-1
    y[i] = 0
    For j=0 to N
    If (i-j ≥ 0)
    If (i-j < M)
       y[i] = y[i] + x[j] * h[i-j]
    End for
End for
```

En la mayoría de las aplicaciones del procesamiento digital de señales, la señal de entrada es del orden de cientos, miles o pocos millones de muestras en longitud, en cambio, la respuesta al impulso es mucho más corta, del orden de pocas muestras a cientos. Por tanto, para nuestra implementación, hemos utilizado que la señal de entrada x[n] está formada por m = 32 muestras, mientras que la respuesta al impulso h[n] por n = 9.

El software utilizado para estos resultados ha sido para los procesadores ARM7 y ARM Cortex-M3 µVision3 V 3.60 (c) Keil Elektronik GmbH / Keil Software, Inc. El software utilizado para sintetizar la operación de convolución en el DSP TMS320C6713 ha sido el proporcionado por Texas Instrument para sus dispositivos. Se trata de Code Composer Studio Version 3.1 de Texas Instrument. Para las FPGA Spartan 3 y 6 la herramienta software Mentor Graphics ModelSim, Xilinx ISE 12.1.

Los dispositivos ARM utilizados concretamente han sido ARM7= LPC2378 a 12MHz y Cortex M3= STM32F103B a 12MHz. El dispositivo DSP ha sido el TMS320C6713B que trabaja a 300 MHz, con velocidades superiores a 1800 millones de operaciones en

punto flotante por segundo (MFLOPS), 2400 millones de instrucciones por segundo (MIPS), y con multiplicadores duales punto fijo/punto flotante del orden de 600 millones de operaciones multiplica–acumula por segundo (MMACS). Con respecto a las FPGAs utilizadas han sido la FPGA de Xilinx Spartan 3 XC3S500e-5 y la Spartan 6 Xilinx XC6SLX100–3FGG676 de velocidad –3.

En la tabla 7-2 se muestran los resultados experimentales obtenidos para tres valores de los bits en que se descompone cada una de las muestras de las señales..

Tabla 7-2 Resultados de la convolución para M=32 y N=9 en los diferentes dispositivos en estudio.

Nº bits	ARM7	ARM Cortex–M3	DSP TMS320C6713	FPGA LUT–4	FPGA LUT–6
16	526 µs	188 µs	53 µs	1,87 µs	0,74 µs
32	423 µs	180 µs	67 µs	4,28 µs	2,44 µs
64	423 µs	180 µs	64 µs	3,21 µs	3,24 µs

Estos resultados se muestran gráficamente de una manera más visual en la figura 7-9.

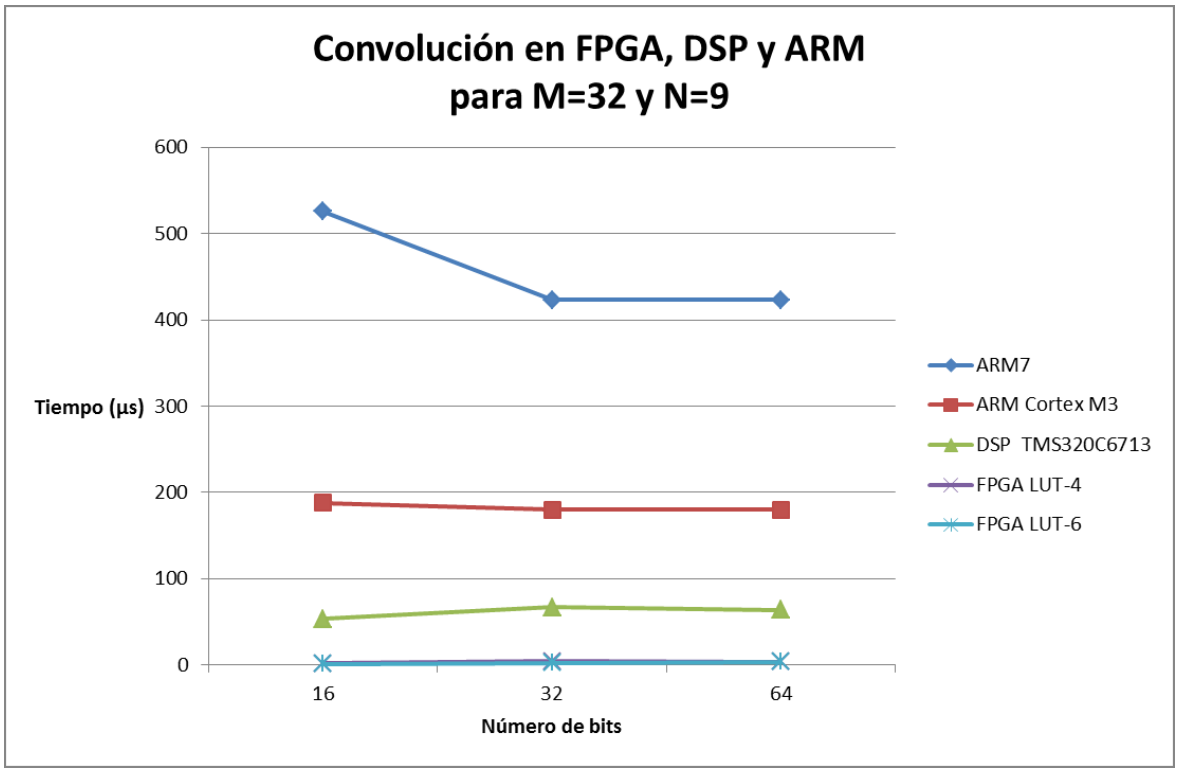


Figura 7-9 Resultados de la operación de convolución en FPGA, DSP y ARM.

La lógica programable disponible en las FPGAs es una alternativa a los procesadores digitales de señal (DSPs) de propósito general, y en ellas se pueden implementar funciones que no es posible implementar en los DSPs actuales. En muchas aplicaciones donde se

requiere procesamiento en tiempo real, imponen considerables restricciones en tamaño físico, consumo de potencia y coste. Otra restricción adicional que sucede con frecuencia es la flexibilidad de la solución, para habilitar la reconfigurabilidad o modificación mientras se está ejecutando.

Algunos procesadores DSPs más potentes (y consecuentemente más caros) tienen múltiples unidades MACs. Estos procesadores realizan múltiples operaciones MACs en un solo ciclo de reloj. El mismo objetivo se consigue utilizando múltiples unidades DSP con un solo MAC con memoria compartida de alta velocidad. En ambos casos, se obtiene un rendimiento superior con componentes de mayor coste.

Los dispositivos FPGA ofrecen una arquitectura que es incluso más potente que la adaptada a una aplicación específica. Debido a la flexibilidad de la lógica incluida en las FPGAs, las funciones DSP pueden ser mapeadas directamente en los recursos de que dispone la FPGA. Las FPGAs, además de ser más rápida la implementación de los algoritmos, ofrecen más ventajas de rendimiento respecto a los componentes utilizados.

El objetivo de esta comparación era realizar un análisis cuantitativo de los tiempos que se tardaría en realizar la operación de convolución en cada uno de estos dispositivos. A priori se supone que los mejores tiempos lo daría la implementación en una FPGA porque optimizará mejor el hardware, seguido del DSPs al ser un procesador especializado en la operación MAC (multiplica y acumula) y por último las dos opciones de ARM. Los resultados obtenidos son coherentes con la suposición inicial.

8 Conclusiones y principales aportaciones

RESUMEN

En este último capítulo se describe un breve resumen de la tesis, se enumeran las conclusiones obtenidas y las principales aportaciones a la investigación. También se comentan los artículos que se han publicado como resultados de esta investigación y se indican brevemente las líneas futuras a seguir como continuación de los estudios realizados.

8.1. Resumen

Una vez concluída la realización de esta tesis doctoral, en este primer apartado del capítulo final voy a describir brevemente el contenido de ésta.

En una primera instancia podría agrupar el contenido de esta memoria en tres grandes bloques. El primero de ellos correspondería a los dos primeros capítulos, en donde el primero se dedica a justificar la realización de la tesis y encuadrarla dentro del procesamiento digital de señales, señalando los objetivos generales de la tesis cuyo fin principal es la optimización de la operación de convolución en dispositivos electrónicos de bajo coste. El segundo capítulo corresponde a una descripción teórica de la operación matemática de la convolución, situándola dentro de los sistemas lineales e invariantes en el tiempo para pasar a explicar su significado físico, sus propiedades matemáticas más importantes así como sus aplicaciones. También se explica su cálculo matemático en tiempo discreto que es el que nos interesará para los datos experimentales de esta tesis.

El segundo bloque de contenidos corresponde a la situación actual de las investigaciones dentro del tema central de esta tesis, o lo que en la literatura actual se conoce como "estado del arte". En este bloque se encuadran los tres siguientes capítulos. El capítulo 3 se dedica a estudiar los algoritmos de suma y multiplicación que más se han utilizado históricamente sobre los dispositivos electrónicos y que más se siguen utilizando

actualmente, describiendo el método y las arquitecturas que se emplean para resolver las operaciones matemáticas básicas como son la suma y la multiplicación, fundamentales para el cálculo de la convolución en tiempo discreto. Este capítulo concluye con la explicación de la aritmética *carry–save* por ser la que se utilizará en el bloque de los datos experimentales como la más idónea para el cálculo de la convolución.

El capítulo 4 se dedica al estudio de los dispositivos electrónicos disponibles en la industria para el procesamiento digital de señales: en su primera parte describe la arquitectura de los procesadores digitales de señal que se utilizarán en los resultados comparativos del capítulo 7, pero principalmente está dedicado a los dispositivos lógicos programables, en particular a las FPGAs y en concreto a la descripción detallada de las FPGA de bajo coste ofrecidas por Xilinx, como son la Spartan 3 y la Spartan 6, por ser las más utilizadas en las investigaciones actuales para el procesamiento digital de señales sin representar un excesivo coste y, por tanto, las utilizadas en la presentación de las arquitecturas que se proponen en esta tesis para optimizar los tiempos de cálculo de la operación de convolución sin aumentar considerablemente el área consumida.

El último capítulo de este bloque del estado de las investigaciones actuales, se enumera en el capítulo 5. En este capítulo se comentan las investigaciones disponibles en la bibliografía sobre la operación de convolución en FPGA, o las relacionadas con ella, tales como los artículos que utilizan la aritmética *carry–save* en sus publicaciones, los que se dedican al estudio de los sumadores y multiplicadores en FPGAs, así como los que utilizan multiplicadores empotrados dentro de las FPGAs o si nos referimos a bloques DSPs integrados dentro de la misma FPGA. Para finalizar el capítulo también se comentan los artículos dedicados a aplicaciones de la convolución, por ejemplo los filtros FIR tan utilizados en el procesamiento digital de señales, o los que centran su estudio sólo en optimizar la unidad multiplica–acumula para este tipo de dispositivos.

El tercer bloque de la memoria corresponde a la presentación de las arquitecturas novedosas sobre FPGA para optimizar el cálculo de la convolución y a la presentación y análisis de los resultados experimentales. En concreto en el capítulo 6 se explica una nueva arquitectura para la convolución cuando los operandos son menores de 48 bits, para la FPGA Spartan 3 de Xilinx con los datos experimentales correspondientes; la arquitectura sobre esta FPGA con el empleo de *buffers* circulares para los datos de entrada o salida, dependiendo del algoritmo utilizado para el cálculo de la convolución. Posteriormente se

proponen distintas arquitecturas para el caso de que los operandos con los que se trabaje superen el ancho de los multiplicadores o bloques DSP integrados en la FPGA. En primer lugar se ha descrito una arquitectura que define un MAC basado en aritmética CSA utilizando compresores 4:2 para el sumador–acumulador y se ofrecen los resultados para las FPGAs de Xilinx Spartan 3 y Spartan 6. A continuación se proponen dos arquitecturas novedosas donde se incluyen multiplicadores y sumadores con salida de doble acarreo utilizando la posibilidad de que dispone la Spartan 6 de un sumador ternario: en la primera de ellas se realiza utilizando el multiplicador con salida CSA y un acumulador con salida de doble acarreo (el acarreo i y el acarreo $i+1$) utilizando compresores 5:3 (Fig 6-19); y en la segunda se utiliza el multiplicador CSA con salida de doble acarreo y el acumulador con salida de doble acarreo con compresores 6:3 (Fig 6-21). En todas las arquitecturas se muestran los resultados obtenidos tanto en velocidad como en área y se compara con la que se obtiene haciendo la descripción VHDL en la herramienta de síntesis. Finaliza este capítulo comparando los resultados obtenidos en la Spartan 6 para los cuatro casos estudiados.

Para terminar el bloque donde se muestran resultados experimentales, se ha incluído el capítulo 7, que consiste en una comparación de tres de los tipos de dispositivos electrónicos de bajo coste: FPGA, DSP y ARM. Las FPGA y los DSP ya se estudiaron en el capítulo 4, por lo que en este se explica la arquitectura del DSP concreto que se utiliza (TMS320C6713 de Texas Instrument) y una descripción de los procesadores ARM utilizados para la comparación (ARM7 y ARM Cortex M3), como ejemplo de procesador de propósito general. Se muestran los resultados experimentales para la operación de convolución en estos tres tipos de dispositivos electrónicos.

8.2. Conclusiones

Una vez visto el resumen de la tesis, se pueden obtener varias conclusiones:

- En primer lugar, constatar que las FPGA son los circuitos electrónicos actuales más utilizados por su gran versatilidad en el desarrollo de aplicaciones para el procesamiento digital de señales y su eficiencia para la implementación de nuevas arquitecturas que permitan implementar las aplicaciones más utilizadas en dicho procesamiento.

- Por otro lado, el uso de la aritmética redundante ha sido denostado en los últimos años en la bibliografía, para su uso en FPGAs debido a que este tipo de aritmética utiliza más recursos hardware. Las aritméticas redundantes típicas son la aritmética *carry–save* y la aritmética *signed—digit*. En el desarrollo de esta tesis se ha demostrado que la utilización de la aritmética *carry–save* es más eficiente cuando el número de operaciones a realizar es muy elevado, como sucede en el cálculo de la operación de convolución.

- En los datos experimentales presentados se demuestra que se pueden obtener resultados análogos en dispositivos FPGA de bajo coste, como es el caso de las de Xilinx Spartan 3 y Spartan 6, que en otros dispositivos más potentes.

- Los resultados se han obtenido para las FPGAs de bajo coste de Xilinx con LUT de 4 entradas y de 6 entradas. Estas mismas conclusiones se podrían aplicar a dispositivos FPGAs de otras compañías, como por ejemplo Altera, que proporcionan circuitos electrónicos con este tipo de arquitectura.

- Finalmente, en el capítulo 7, se ha comprobado tabularmente, que los mejores resultados se obtienen en FPGAs de bajo coste frente a los tradicionales procesadores de propósito general (ARM7 y ARM Cortex M3) y los procesadores digitales de señal (DSP TMS320C6713 de Texas Instrument.

8.3. Principales aportaciones

La principal aportación de esta tesis consiste en la utilización de la aritmética carry–save para el procesamiento digital de señales en los dispositivos FPGA. Como se ha comentado anteriormente, el uso de aritmética redundante (CS y SD) ha estado en desuso en las investigaciones de la última década debido a que consumía recursos que no se correspondían con los resultados esperados. Con los cálculos experimentales de esta tesis se ha comprobado que, cuando el número de operaciones MAC es elevado, como sucede en el caso de la convolución de dos señales, el uso de la aritmética *carry–save* está justificado debido a que el aumento de área es pequeño comparado con las ventajas que se obtienen en la disminución del retardo.

Se han introducido varias arquitecturas nuevas que optimizan los resultados para la operación MAC en los dispositivos FPGA de bajo coste, en particular las FPGA de Xilinx

Spartan 3 (LUT 4) y Spartan 6 (LUT 6). En primer lugar, para operandos con resultado menor de 48 bits, se ha introducido una arquitectura con un acumulador CSA–CPA combinado que utiliza pocos recursos más de área dentro de la FPGA obteniendo mejores resultados temporales, en una FPGA Spartan 3. En este mismo dispositivo también se definieron dos arquitecturas nuevas para los operandos de entrada y salida utilizando los algoritmos desde el punto de vista de la entrada y desde el punto de vista de la salida, introducuendo los *buffers* circulares en la FPGA. Los buffers circulares son ampliamente utilizados en los procesadores digitales de señal, ya que son procesadores específicos para el procesamiento digital de la señal y necesitan tener entre sus modos de direccionamiento el direccionamiento circular, para implementar algoritmos como la FFT y la convolución.

Finalmente, también se han definido tres nuevas arquitecturas para realizar la operación de convolución cuando los resultados del acumulador son mayores de 48 bits, utilizando los compresores, en concreto los compresores 4:2, compresores 5:3 y compresores 6:3, optimizando al máximo la posibilidad de las FPGA que tienen LUT de 6 entradas, en nuestro caso concreto la Spartan 6 de Xilinx. En estas tres arquitecturas se utiliza la aritmética *carry–save*: en la primera el multiplicador da salida CSA y para el acumulador se emplea un compresor 4:2 (Fig 6-11). En la segunda el multiplicador proporciona salida *carry–save* y se utiliza para el acumulador un sumador CSA de doble acarreo y un compresor 5:3 (Fig 6-19). Finalmente, la tercera de estas arquitecturas muestra un operador MAC, donde el multiplicador muestra su resultado en CSA de doble acarreo y el sumador–acumulador también proporciona salida de doble acarreo. En estas dos últimas se utiliza la posibilidad que tienen estas FPGA de configurar los sumadores como ternarios.

8.4. Publicaciones

En este apartado comentaré brevemente seis de las publicaciones relacionadas con el objeto de esta tesis. En primer lugar comentaré los tres artículos que están directamente relacionados con el contenido, y de los que soy primer autor, y a continuación otros tres, realizados con el grupo de investigación que, aunque no están aplicados en la operación de convolución, sí se aplican sobre las FPGAs que se han utilizado en esta tesis doctoral.

El primer trabajo que se realizó sobre estos estudios fue el siguiente [94]: *Efficient mapping on FPGA of convolution computation based on combined CSA–CPA*

accumulator. Moreno, C. D.; Quiles, F. J.; Ortiz, M. A.; Brox, M.; Hormigo, J.; Villalba, J.; Zapata, E. L. ICECS 2009. 16th IEEE International Conference on Electronics, Circuits, and Systems, 2009. Yasmine Hammamet, Tunisia. Pages 419-422. 13-16 Dec. 2009. D.O.I.: 10.1109/ICECS.2009.5410903. INSPEC Accession Number: 11142053. Los resultados de este artículo, expuestos en el capítulo 6 (apartado 6.2), corresponden a la propuesta de una arquitectura novedosa de un acumulador combinado CSA–CPA en la Spartan 3 para la optimización de los recursos hardware en la operación de convolución. Este congreso está indexado en el Conference Ranking. Por otro lado, este artículo ha sido citado en otras investigaciones, tales como Hashemi y Eshghi [46], lo cual comprueba la actualidad del tema de esta tesis en la investigación actual.

Un segundo trabajo publicado corresponde al estudio de los buffers circulares presentando una nueva arquitectura y sus resultados corresponden al artículo [47], que han sido expuestos en el capítulo 2 y el capítulo 6 de esta memoria. La referencia del artículo es la siguiente: *Convolution Computation in FPGA Based on Carry-Save Adders and Circular Buffers*. Moreno, Carlos D.; Martínez, Pilar; Bellido, Francisco J.; Hormigo, Javier; Ortiz, Manuel A.; Quiles, Francisco J. IT Revolutions. Springer Berlin Heidelberg. D.O.I.: http://dx.doi.org/10.1007/978-3-642-32304-1_20. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Volume 82. Chapter 20. Pages 237-248. ISBN 978-3-642-32303-4. 2012. Los resultados de este estudio se realizaron sobre la Spartan 3 y están expuestos en el apartado 6.3 de esta memoria de tesis doctoral. Este trabajo también está indexado en el Conference Ranking.

El tercer trabajo sobre el tema de la tesis corresponde al capítulo 7 y consistió en una ponencia en un congreso [96]. *Comparación de la operación de convolución en diferentes dispositivos electrónicos de bajo coste*. Moreno Moreno, Carlos Diego; Martínez Jiménez, Pilar; Bellido Outeiriño, Francisco José; Hormigo Aguilar, Francisco Javier. III Congreso Científico de Investigadores en Formación de la Universidad de Córdoba. Córdoba, 9 y 10 de Abril de 2013. Escuela Internacional de Doctorado del Campus de Excelencia Internacional en Agroalimentación (ceiA3, eidA3) y Escuela Multidisciplinar de Doctorado de la UCO (ED-UCO).

A continuación se relatan otras tres publicaciones, sobre estudios realizados sobre FPGAs, aunque no estudian directamente la operación de la convolución. En primer lugar se nombra el artículo siguiente [109]: *Acelerador Hardware de bajo coste para bus PCI*

Convencional. Francisco J. Quiles, Manuel Ortiz, Miguel A. Montijano, Carlos D. Moreno, María Brox, Javier Hormigo, Julio Villalba. Seminario Anual de Automática, Electrónica Industrial e Instrumentación 2012 (SAAEI'12). Guimarães, Portugal. 11, 12 y 13 de julio de 2012. En este trabajo se presenta un acelerador hardware de bajo coste para el bus PCI convencional. El acelerador está formado por dos partes: un interfaz al bus PCI y una FPGA para la implementación del algoritmo del acelerador. La simplicidad de este acelerador hardware permite que el diseñador dedique su esfuerzo al algoritmo que implementará en la FPGA mientras que el esfuerzo que dedica al interfaz PCI es mínimo ya que el interfaz al bus PCI se ha reducido al máximo. El *bridge* con el bus PCI convencional está implementado en un core IP, que permite la programación dinámica de la FPGA e implementa facilidades para el control de este recurso por el sistema operativo.

Un segundo trabajo a nombrar sobre FPGA corresponde al siguiente [110]: *UCORE: Reconfigurable Platform for Educational Purposes*. Quiles, F. J.; Ortiz, M.; Brox, M.; Moreno, C. D.; Hormigo, J.; Villalba, J. Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on. IEEE Computer Society. Pages 109-114. 13-15 December 2010, Cancun, Quintana Roo, Mexico. D.O.I.: 10.1109/ReConFig.2010.60. ISBN: 978-0-7695-4314-7. INSPEC Accession Number: 11772008. Este congreso está indexado en el Conference Ranking. En este artículo se presentó una plataforma didáctica para realizar prácticas en sistemas digitales con un interfaz al bus PCI convencional, basada en un hardware reconfigurable muy útil para el diseño de aceleradores hardware y sistemas que necesiten interfaz al bus PCI. La plataforma consiste de un componente hardware y software que permite desarrollar fácilmente prototipos de sistemas electrónicos en general, sistemas simples con un interfaz convencional al bus PCI, aceleradores hardware y buses. Para el hardware hemos desarrollado una placa de prototipo rápido llamada UCOS3E5 y para el software una aplicación llamada PCI Bus Handler. Esta plataforma ha sido muy útil para los resultados experimentales de esta investigación.

Por último voy a reseñar un artículo [111] también relacionado con el tema de FPGA: *Using soft processors for component design in SOC: A case-study of timers*. Ortiz, M.; Brox, M.; Quiles, F.; Gersnoviez, A.; Moreno, C.; Montijano, M. System-on-Chip, 2008. SOC 2008. International Symposium on. IEEE. Pages 1-4. Tampere, Finland. 5-6 Nov. 2008. D.O.I.: 10.1109/ISSOC.2008.4694873. ISBN: 978-1-4244-2541-9. INSPEC

Accession Number: 10411531. Este congreso también aparece indexado en el Conference Ranking. En este trabajo se presentaba el desarrollo de una aplicación de codiseño hardware/software de *system on chip*, que consistía en el desarrollo de temporizadores hardware/software utilizando procesadores software, tales como el *picoblaze*, en la FPGA Spartan 3 de Xilinx.

8.5. Trabajos futuros

Una de las aportaciones principales de esta tesis doctoral consiste en la optimización de la unidad multiplica–acumula para la operación de convolución en dispositivos electrónicos de bajo coste, básicamente FPGAs. Por tanto, cualquier operación dentro del procesamiento digital de señales que tenga que realizar este cálculo puede ser optimizado, así como las aplicaciones de la operación de convolución.

Como aplicaciones directas de la operación de convolución se pueden reseñar la realización de filtros FIR, que consiste en una operación de convolución donde una de las señales son los coeficientes del filtro y la otra la señal que se desea filtrar. Otro tipo de filtros digitales, tales como los filtros IIR también se podrían optimizar. Dentro del campo del procesamiento digital de señales existen infinidad de aplicaciones de los filtros digitales, como pueden ser aplicaciones de gráficos (JPEG), de audio (MP3), de video (MPEG), etc. Por ejemplo, para las aplicaciones de imágenes se utiliza la convolución en dos dimensiones, que sería una de los posibles trabajos a desarrollar como aplicación de esta investigación: la convolución 2D.

Otros aplicaciones dentro del procesamiento digital de señales que tienen como base la operación MAC son la correlación, la transformada rápida de Fourier (FFT), la transformada discreta del coseno, etc. Habría que seguir investigando sobre los resultados de la optimización de la operación MAC para este tipo de operaciones.

9 Bibliografía

[1] H. Nyquist, "Certain Topics in Telegraph Transmission Theory," *American Institute of Electrical Engineers, Transactions of the,* vol. 47, pp. 617-644, 1928.

[2] C. E. Shannon, "Communication in the Presence of Noise," *Proceedings of the IRE,* vol. 37, pp. 10-21, 1949.

[3] J. W. Cooley, and Tukey, J. W, "An algorithm for the Machine Computation of Complex Fourier Series," *Mathematics of Computations,* vol. 19, pp. 297-301, April 1965.

[4] D. G. M. John G. Proakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Third Edition ed.: Prentice Hall International, INC, 1996.

[5] S. W. Smith, *Digital Signal Processing,a Practical Guide for Engineers and Scientists*: Elsevier Science, 2003.

[6] L. M. Jiménez García, Fernández Peris, César, Puerto Manchón, Rafael, García Aracil, Nicolás Manuel, Sabater Navarro, José María, Torres Medina, Fernando, "Control de enfoque de un sistema activo de visión estereoscópica," *Memorias del X Simposio de Ingeniería Eléctrica (SIE'01). Santa Clara (Cuba),* p. 6, 2001.

[7] R. J. Read. (2009, Marzo 2013). *The convolution theorem and its applications* Available: http://www-structmed.cimr.cam.ac.uk/Course/Convolution/convolution.html

[8] F. J. C. R. Miguel Mora González, Jesús Muñoz Maciel, Julio C. Martínez Romo, and C. A. d. L. O. Francisco J. Luna Rosas, Gilberto Gómez Rosas, F. Gerardo Peña Lecona. (2008, ENERO-ABRIL 2008) Reducción de ruido digital en señales ECG utilizando filtraje por convolución. *Investigación y Ciencia*, . 7. Available: http://www.redalyc.org/articulo.oa?id=67404005

[9] A. F. Sadi Bin. (2008, October 22. 2008) Development of a Stochastic Model to Evaluate Economic Capital for Credit Concentration Risk. 69.

[10] M. D. E. T. Lang, *Digital Arithmetic*, 2003.

[11] J. L. Beuchat and J. M. Muller, "Automatic Generation of Modular Multipliers for FPGA Applications," *IEEE Transactions on Computers,* vol. 57, pp. 1600-1613, 2008.

[12] C. S. Wallace, "A Suggestion for a Fast Multiplier," *Electronic Computers, IEEE Transactions on,* vol. EC-13, pp. 14-17, 1964.

[13] J. A. a. R. L. S. Rajchman, "An Electrically-Focused Multiplier Phototube," *Electronics,* pp. 20-23, 1940.

[14] A. D. Booth, "A Signed Binary Multiplication Technique," *The Quarterly Journal of Mechanics and Applied Mathematics,* vol. 4, pp. 236-240, January 1, 1951 1951.

[15] L. P. Rubinfield, "A Proof of the Modified Booth's Algorithm for Multiplication," *Computers, IEEE Transactions on,* vol. C-24, pp. 1014-1015, 1975.

[16] H. Sam and A. Gupta, "A generalized multibit recoding of two's complement binary numbers and its proof with application in multiplier implementations," *IEEE Transactions on Computers,* vol. 39, pp. 1006-1015, 1990.

[17] A. R. Cooper, "Parallel architecture modified Booth multiplier," *Electronic Circuits and Systems, IEEE Proceedings G,* vol. 135, pp. 125-128, 1988.

[18] M. R. Santoro and M. A. Horowitz, "SPIM: a pipelined 64×64-bit iterative multiplier," *Solid-State Circuits, IEEE Journal of,* vol. 24, pp. 487-493, 1989.

[19] L. Dadda. (1965, 29 de abril de 1965) Some schemes for parallel multipliers. *Alta Frecuenza*. 349–356.

[20] P. R. Cappello and K. Steiglitz, "A VLSI layout for a pipelined Dadda multiplier," *ACM Trans. Comput. Syst.,* vol. 1, pp. 157-174, 1983.

[21] A. Dhurkadas, "Faster parallel multiplier," *Proceedings of the IEEE,* vol. 72, pp. 134-136, 1984.

[22] E. J. y. J. M. E.I. Boemo, "Taxonomía de Multiplicadores," presented at the VIII DCIS Conference, Design of Circuit and Integrated System, 1993.

[23] D. F. Denis Teixeira, A. Susim and L. Carro, "Comparaçao de multiplicadores em FPGA," in *V Workshop Iberchip,* Lima, Perú, 1999, pp. 182-187.

[24] O. Nibouche, A. Bouridane, and M. Nibouche, "New architectures for serial-serial multiplication," in *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on,* 2001, pp. 705-708 vol. 2.

[25] A. Aggoun, A. Ashur, and M. K. Ibrahim, "Area-time efficient serial-serial multipliers," in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on,* 2000, pp. 585-588 vol.5.

[26] A. Aggoun, A. F. Farwan, M. K. Ibrahim, and A. Ashur, "Radix-2n serial-serial multipliers," *Circuits, Devices and Systems, IEE Proceedings -*, vol. 151, pp. 503-509, 2004.

[27] A. Bouridane, M. Nibouche, O. Nibouche, D. Crookes, and B. Albesher, "A low latency bi-directional serial-parallel multiplier architecture," in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, 2000, pp. 593-596 vol.5.

[28] N. Shirazi, A. Walters, and P. Athanas, "Quantitative analysis of floating point arithmetic on FPGA based custom computing machines," in *FPGAs for Custom Computing Machines, 1995. Proceedings. IEEE Symposium on*, 1995, pp. 155-162.

[29] W. B. Ligon, III, S. McMillan, G. Monn, K. Schoonover, F. Stivers, and K. D. Underwood, "A re-evaluation of the practicality of floating-point operations on FPGAs," in *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*, 1998, pp. 206-215.

[30] J. L. H. D. A. Patterson, *Computer architecture: a quantitative approach. 5th Edition*, 5th Edition ed.: Morgan Kaufmann, 2011.

[31] J. P. Hayes, *Introducción al diseño lógico digital*: Addison-Wesley, 1996.

[32] L. A. Gustavo E. Ordóñez Fernández and J. V. M. López López, "Diseño de Multiplicadores Paralelos de 16 bits en FPGAs," presented at the X Workshop IBERCHIP, Cartagena, Colombia, 2004.

[33] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York, Oxford: Oxford University Press, 2010.

[34] H. H. Guild, "Fully iterative fast array for binary multiplication and addition," *Electronics Letters*, vol. 5, p. 263, June 12, 1969 1969.

[35] J. V. McCanny and J. G. McWhirter, "Completely iterative, pipelined multiplier array suitable for VLSI," *Electronic Circuits and Systems, IEE Proceedings G*, vol. 129, pp. 40-46, 1982.

[36] M. Ortiz, F. Quiles, J. Hormigo, F. J. Jaime, J. Villalba, and E. L. Zapata, "Efficient Implementation of Carry-Save Adders in FPGAs," in *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, 2009, pp. 207-210.

[37] W.-S. G. Sen-Maw Kuo, *Digital signal processors: architectures, implementations, and applications*: Pearson Prentice Hall, 2005.

[38] Y. H. Hu, *Programmable Digital Signal Processors: Architecture, Programming and Applications*: CRC Press, 2001.

[39] G. Frantz, "DSP's Past Can't Hold A Candle to its Future," vol. 2013, Prsync, Ed., ed: Texas Instrument, 2007.

[40] P. B. Lapsley, J.; Shoham, A.; Lee, E. , *DSP Processor Fundamentals:Architectures and Features*: Wiley-IEEE Press 1997.

[41] Xilinx. (2011). *Spartan-3 Generation FPGA User Guide (UG331 (v1.8) ed.)*. Available: http://www.xilinx.com/support/documentation/spartan-3.htm.

[42] Xilinx. (2008). *XtremeDSP DSP48A for Spartan-3A DSP FPGAs. User Guide (UG431 (v1.3) ed.).* Available: http://www.xilinx.com/support/documentation/spartan-3.htm

[43] Xilinx. (2010). *Spartan-6 FPGA Configurable Logic Block. User Guide (UG384 (v1.1) ed.).* Available: http://www.xilinx.com/support/documentation/spartan-6.htm.

[44] Xilinx. (2009). *Spartan-6 FPGA DSP48A1 Slice. User Guide. (UG389 (v1.1) ed.)*. Available: http://www.xilinx.com/products/spartan6

[45] I. Berkeley Design Technology, "FPGAs for DSP," Berkeley Design Technology, Inc. , North California Blvd.2006.

[46] M. R. Hashemi and M. Eshghi, "Design of a reconfigurable parallel convolver," in *Systems, Signals and Image Processing (IWSSIP), 2012 19th International Conference on*, 2012, pp. 181-184.

[47] C. D. Moreno, P. Martínez, F. Bellido, J. Hormigo, M. Ortiz, and F. Quiles, "Convolution Computation in FPGA Based on Carry-Save Adders and Circular Buffers," in *IT Revolutions*. vol. 82, M. Liñán Reyes, J. Flores Arias, J. González de la Rosa, J. Langer, F. Bellido Outeiriño, and A. Moreno-Muñoz, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 237-248.

[48] K. Mohammad and S. Agaian, "Efficient FPGA implementation of convolution," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, 2009, pp. 3478-3483.

[49] Z. Wang, M. Gao, X. Fu, and C. Jiang, "Design of Real-time Convolution Processor and its Application in Radar Echo Signal Simulator," in *Computer Science and Information Technology, 2008. ICCSIT '08. International Conference on*, 2008, pp. 162-166.

[50] E. Jamro and K. Wiatr, "Genetic programming in FPGA implementation of addition as a part of the convolution," in *Digital Systems Design, 2001. Proceedings. Euromicro Symposium on*, 2001, pp. 466-473.

[51] T. Oelsner, "Implementation of Data Convolution Algorithms in FPGA," ed: QuickLogic Europe (QAN18), pp. 5-193, 5-203.

[52] L. M. Garcés Socarrás, A. J. Cabrera Sarmiento, S. Sánchez Solano, and P. Brox Jiménez, *Diseño de bloques de convolución para procesado de imágenes con FPGA* vol. 32, 2011.

[53] Jagadguru Swami Sri Bharati Krsna Tirthaji Maharaja, *Vedic Mathematics or Sixteen simple Mathematical Formulae from the Vedas*: QMOT1LA. LBANARSIDASS Indologlcal Publishers & Booksellers, 1965.

[54] M. R. R. Kulkarni, "Parallel Hardware Implementation of Convolution using Vedic Mathematics," *IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)*, vol. 1, pp. 21-26, Nov. - Dec. 2012 2012.

[55] A. Haveliya, "FPGA Implementation of a Vedic Convolution Algorithm," *International Journal of Engineering Research and Applications (IJERA)*, vol. 2, pp. 678-684, Jan-Feb 2012 2012.

[56] A. Verma, A. K. Verma, H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Synthesis of Floating-Point Addition Clusters on FPGAs Using Carry-Save Arithmetic," in *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, 2010, pp. 19-24.

[57] R. Gutierrez, J. Valls, and A. Perez-Pascual, "FPGA-implementation of Time-Multiplexed Multiple Constant Multiplication based on carry-save arithmetic," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009, pp. 609-612.

[58] M. A. Erle, M. J. Schulte, and B. J. Hickmann, "Decimal Floating-Point Multiplication Via Carry-Save Addition," in *Computer Arithmetic, 2007. ARITH '07. 18th IEEE Symposium on*, 2007, pp. 46-55.

[59] O. Gustafsson, A. G. Dempster, and L. Wanhammar, "Multiplier blocks using carry-save adders," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, 2004, pp. II-473-6 Vol.2.

[60] J. Hormigo, J. Villalba, and E. L. Zapata, "Multi-operand Redundant Adders on FPGAs," *IEEE Transactions on Computers*, vol. PP, pp. 1-1, 2012.

[61] D. H. K. Hoe, C. Martinez, and S. J. Vundavalli, "Design and characterization of parallel prefix adders using FPGAs," in *System Theory (SSST), 2011 IEEE 43rd Southeastern Symposium on*, 2011, pp. 168-172.

[62] S. Bhattacharjee, S. Sil, B. Basak, and A. Chakrabarti, "Evaluation of power efficient adder and multiplier circuits for FPGA based DSP applications," in *Communication and Industrial Application (ICCIA), 2011 International Conference on*, 2011, pp. 1-5.

[63] S. Gao, D. Al-Khalili, and N. Chabini, "Efficient Realization of Large Size Two's Complement Multipliers Using Embedded Blocks in FPGAs," *Circuits, Systems & Signal Processing*, vol. 27, pp. 713-731, 2008/10/01 2008.

[64] C. R. W. Baugh, B.A. , "A Two's Complement Parallel Array Multiplication Algorithm," *Computers, IEEE Transactions on*, vol. C-22, pp. 1045 - 1047 Dec. 1973 1973.

[65] G. Shuli, D. Al-Khalili, and N. Chabini, "Optimized realization of large-size two's complement multipliers on FPGAs," in *Circuits and Systems, 2007. NEWCAS 2007. IEEE Northeast Workshop on*, 2007, pp. 494-497.

[66] G. Shuli, N. Chabini, and D. Al-Khalili, "256x256-bit multiplier using multi-granular embedded DSP blocks in FPGAs," in *Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA 2008. 2008 Joint 6th International IEEE Northeast Workshop on*, 2008, pp. 253-256.

[67] P. Brisk, A. K. Verma, P. Ienne, and H. Parandeh-Afshar, "Enhancing FPGA performance for arithmetic circuits," presented at the Proceedings of the 44th annual Design Automation Conference, San Diego, California, 2007.

[68] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient synthesis of compressor trees on FPGAs," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, 2008, pp. 138-143.

[69] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Improving Synthesis of Compressor Trees on FPGAs via Integer Linear Programming," in *Design, Automation and Test in Europe, 2008. DATE '08*, 2008, pp. 1256-1261.

[70] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, pp. 203-215, 2007.

[71] W. J. Stenzel, W. J. Kubitz, and G. H. Garcia, "A Compact High-Speed Parallel Multiplication Scheme," *Computers, IEEE Transactions on*, vol. C-26, pp. 948-957, 1977.

[72] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Exploiting fast carry-chains of FPGAs for designing compressor trees," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009, pp. 242-249.

[73] H. Parandeh-Afshar, A. Cevrero, P. Athanasopoulos, P. Brisk, Y. Leblebici, and P. Ienne, "A flexible DSP block to enhance FPGA arithmetic performance," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, 2009, pp. 70-77.

[74] H. Parandeh-Afshar and P. Ienne, "Highly Versatile DSP Blocks for Improved FPGA Arithmetic Performance," in *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, 2010, pp. 229-236.

[75] H. Parandeh-Afshar, A. K. Verma, P. Brisk, and P. Ienne, "Improving FPGA Performance for Carry-Save Arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, pp. 578-590, 2010.

[76] H. Parandeh-Afshar and P. Ienne, "Measuring and Reducing the Performance Gap between Embedded and Soft Multipliers on FPGAs," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, 2011, pp. 225-231.

[77] O. Kwon, K. Nowka, and J. E. E. Swartzlander, "A 16-bit x 16-bit MAC design using fast 5:2 compressor," *Proceedings of the IEEE International Conference on Application Specific Systems, Architectures, and Processors*, pp. 235-243, 2000.

[78] O. Kwon, K. Nowka, and J. E. E. Swartzlander, "A 16-Bit by 16-Bit MAC Design Using Fast 5: 3 Compressor Cells," *The Journal of VLSI Signal Processing*, vol. 31, pp. 77-89, 2002.

[79] M. Rawski, P. Tomaszewicz, H. Selvaraj, and T. Luba, "Efficient Implementation of digital filters with use of advanced synthesis methods targeted FPGA architectures," in *Digital System Design, 2005. Proceedings. 8th Euromicro Conference on*, 2005, pp. 460-466.

[80] M. Rawski, P. Tomaszewicz, B. J. Falkowski, and T. Łuba, "Application of Advanced Logic Synthesis in FPGA-based Implementations of Digital Filters," presented at the DASIP'2007 - Design & Architectures for Signal and Image Processing, France, 2007.

[81] M. R. Staworko, M. Politechnika Warszawska, Instytut Telekomunikacji, "Application of Modified Distributed Arithmetic Concept in FIR Filter Implementations Targeted at Heterogeneous FPGAs " *Przegląd Elektrotechniczny,* vol. R. 88, nr 6, pp. 240--246 2012.

[82] U. Meyer-Baese, C. Jiajia, C. Chip Hong, and A. G. Dempster, "A Comparison of Pipelined RAG-n and DA FPGA-based Multiplierless Filters," in *Circuits and Systems, 2006. APCCAS 2006. IEEE Asia Pacific Conference on*, 2006, pp. 1555-1558.

[83] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, pp. 569-577, 1995.

[84] T. Sasao, Y. Iguchi, and T. Suzuki, "On LUT cascade realizations of FIR filters," in *Digital System Design, 2005. Proceedings. 8th Euromicro Conference on*, 2005, pp. 467-474.

[85] S. Mirzaei, A. Hosangadi, and R. Kastner, "FPGA Implementation of High Speed FIR Filters Using Add and Shift Method," in *Computer Design, 2006. ICCD 2006. International Conference on*, 2006, pp. 308-313.

[86] J. J. Martinez, F. J. Toledo, E. Javier Garrigos, and J. Manuel Ferrández, "FPGA implementation of an area-time efficient FIR filter core using a self-clocked approach," in *Field Programmable Logic and Applications, 2005. International Conference on*, 2005, pp. 547-550.

[87] V. Gierenz, C. Panis, and J. Nurmi, "Parameterized MAC unit generation for a scalable embedded DSP core," in *NORCHIP, 2008.*, 2008, pp. 127-132.

[88] V. Gierenz, C. Panis, and J. Nurmi, "Parameterized MAC unit generation for a scalable embedded DSP core," *Microprocessors and Microsystems*, vol. 34, pp. 138-150, 2010.

[89] L. Yuyun, D. Roberts, and E. Hoffman, "VLSI implementation of a high performance and low power 32-bit multiply-accumulate unit," in *Solid-State Circuits Conference, 2001. ESSCIRC 2001. Proceedings of the 27th European*, 2001, pp. 269-272.

[90] D. Tan, A. Danysh, and M. Liebelt, "Multiple-precision fixed-point vector multiply-accumulator using shared segmentation," in *Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium on*, 2003, pp. 12-19.

[91] E. Jamro and K. Wiatr, "FPGA implementation of addition as a part of the convolution," in *Digital Systems Design, 2001. Proceedings. Euromicro Symposium on*, 2001, pp. 458-465.

[92] V. Sriram and D. Kearney, "A FPGA Implementation of Variable Kernel Convolution," in *Parallel and Distributed Computing, Applications and Technologies, 2007. PDCAT '07. Eighth International Conference on*, 2007, pp. 105-110.

[93] U. Junhyung, K. Taewhan, and C. L. Liu, "Optimal allocation of carry-save-adders in arithmetic optimization," in *Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on*, 1999, pp. 410-413.

[94] C. D. Moreno, F. J. Quiles, M. A. Ortiz, M. Brox, J. Hormigo, J. Villalba, *et al.*, "Efficient mapping on FPGA of convolution computation based on combined CSA-CPA accumulator," in *Electronics, Circuits, and Systems, 2009. ICECS 2009. 16th IEEE International Conference on*, 2009, pp. 419-422.

[95] F. Q. M.A. Ortiz, F. Bellido, J. Hormigo, J. Villalba, "Multipliers with Carry-Save Output on FPGA," Universidad de Córdoba, Internal report2012.

[96] C. D. M. Moreno, "Comparación de la operación de convolución en diferentes dispositivos electrónicos de bajo coste," ed. Salón de actos del Rectorado de la Universidad de Córdoba: Escuela Internacional de Doctorado del Campus de Excelencia Internacional en Agroalimentación (ceiA3, eidA3) y Escuela Multidisciplinar de Doctorado de la UCO (ED-UCO), 2013.

[97] D. C. M. Bilsby, R. L. Walke, and R. W. M. Smith, "Comparison of a programmable DSP and a FPGA for real-time multiscale convolution," in *High Performance Architectures for Real-Time Image Processing (Ref. No. 1998/197), IEE Colloquium on*, 1998, pp. 4/1-4/6.

[98] R. Duren, J. Stevenson, and M. Thompson, "A comparison of FPGA and DSP development environments and performance for acoustic array processing," in *Circuits and Systems, 2007. MWSCAS 2007. 50th Midwest Symposium on*, 2007, pp. 1177-1180.

[99] M. Montani, L. De Marchi, A. Marcianesi, and N. Speciale, "Comparison of a programmable DSP and FPGA implementation for a wavelet-based denoising algorithm," in *Circuits and Systems, 2003 IEEE 46th Midwest Symposium on*, 2003, pp. 602-605 Vol. 2.

[100] K. Wiatr, "Median and Morphological Specialized Processors for a Real-Time Image Data Processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2002, pp. 115-121, 2002.

[101] K. Underwood, "FPGAs vs. CPUs: trends in peak floating-point performance," presented at the Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, Monterey, California, USA, 2004.

[102] B. Cope, "Implementation of 2D Convolution on FPGA, GPU and CPU," Department of Electrical & Electronic Engineering, Imperial College London, 2006.

[103] D. B. Thomas, L. Howes, and W. Luk, "A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation," presented at the Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, Monterey, California, USA, 2009.

[104] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009, pp. 126-131.

[105] T. Instrument. (2005, November 2005). TMS320C6713, Floating-Point Digital Signal Processor. Available: http://www.ti.com/product/tms320c6713

[106] A. ltd. (2012). *ARM: The architecture of digital world*. Available: http://www.arm.com/products/processors/

[107] T. Martin, *The Insider's Guide To The Philips ARM7 Based Microcontrollers. An Engineer's Introduction To The LPC2100 Series*: Hitex (UK) Ltd., 2005.

[108] J. Yiu, *The Definitive Guide to the ARM Cortex-M3*: Newnes, 2009.

[109] M. O. Francisco J. Quiles, Miguel A. Montijano, Carlos D. Moreno, María Brox, Javier Hormigo, Julio Villalba, "Acelerador hardware de bajo coste para bus PCI convencional," presented at the Seminario Anual de Automática, Electrónica Industrial e Instrumentación 2012 (SAAEI'12), Guimarães, Portugal, 2012.

[110] F. J. Quiles, M. Ortiz, M. Brox, C. D. Moreno, J. Hormigo, and J. Villalba, "UCORE: Reconfigurable Platform for Educational Purposes," in *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, 2010, pp. 109-114.

[111] M. Ortiz, M. Brox, F. Quiles, A. Gersnoviez, C. Moreno, and M. Montijano, "Using soft processors for component design in SOC: A case-study of timers," in *System-on-Chip, 2008. SOC 2008. International Symposium on*, 2008, pp. 1-4.

Anexo A: Publicaciones

Publicaciones resultado de la investigación

- ***Efficient mapping on FPGA of convolution computation based on combined CSA–CPA accumulator***. Moreno, C. D.; Quiles, F. J.; Ortiz, M. A.; Brox, M.; Hormigo, J.; Villalba, J.; Zapata, E. L. ICECS 2009. 16th IEEE International Conference on Electronics, Circuits, and Systems, 2009. Yasmine Hammamet, Tunisia. Pages 419-422. 13-16 Dec. 2009. D.O.I.: 10.1109/ICECS.2009.5410903. INSPEC Accession Number: 11142053.

- ***Convolution Computation in FPGA Based on Carry-Save Adders and Circular Buffers***. Moreno, Carlos D.; Martínez, Pilar; Bellido, Francisco J.; Hormigo, Javier; Ortiz, Manuel A.; Quiles, Francisco J. IT Revolutions. Springer Berlin Heidelberg. D.O.I.: http://dx.doi.org/10.1007/978-3-642-32304-1_20. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Volume 82. Chapter 20. Pages 237-248. ISBN 978-3-642-32303-4. 2012.

- ***Comparación de la operación de convolución en diferentes dispositivos electrónicos de bajo coste.*** Moreno Moreno, Carlos Diego; Martínez Jiménez, Pilar; Bellido Outeiriño, Francisco José; Hormigo Aguilar, Francisco Javier. III Congreso Científico de Investigadores en Formación de la Universidad de Córdoba. Córdoba, 9 y 10 de Abril de 2013. Escuela Internacional de Doctorado del Campus de Excelencia Internacional en Agroalimentación (ceiA3, eidA3) y Escuela Multidisciplinar de Doctorado de la UCO (ED-UCO).

Otras publicaciones en FPGAs:

- ***Acelerador Hardware de bajo coste para bus PCI Convencional***. Francisco J. Quiles, Manuel Ortiz, Miguel A. Montijano, Carlos D. Moreno, María Brox, Javier Hormigo, Julio Villalba. Seminario Anual de Automática, Electrónica Industrial e Instrumentación 2012 (SAAEI'12). Guimarães, Portugal. 11, 12 y 13 de julio de 2012.

- ***UCORE: Reconfigurable Platform for Educational Purposes***. Quiles, F. J.; Ortiz, M.; Brox, M.; Moreno, C. D.; Hormigo, J.; Villalba, J. Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on. IEEE Computer Society. Pages 109-114. 13-15 December 2010, Cancun, Quintana Roo, Mexico. D.O.I.: 10.1109/ReConFig.2010.60. ISBN: 978-0-7695-4314-7. INSPEC Accession Number: 11772008.

- ***Using soft processors for component design in SOC: A case-study of timers***. Ortiz, M.; Brox, M.; Quiles, F.; Gersnoviez, A.; Moreno, C.; Montijano, M. System-on-Chip, 2008. SOC 2008. International Symposium on. IEEE. Pages 113-116. Tampere, Finland. 5-6 Nov. 2008. D.O.I.: 10.1109/ISSOC.2008.4694873. ISBN: 978-1-4244-2541-9. INSPEC Accession Number: 10411531. Citado en WoK, Accession Number:WOS:000262647700031.

Efficient Mapping on FPGA of Convolution Computation based on Combined CSA-CPA Accumulator

C.D. Moreno, F. J. Quiles, M.A. Ortiz, M. Brox
Department of Computer Architecture
University of Cordoba, SPAIN
Email: el1orlom,el1qulaf@uco.es

J. Hormigo, J. Villalba, E.L. Zapata
Department of Computer Architecture
University of Malaga, SPAIN
Email: hormigo,julio@ac.uma.es

***Abstract*— In this paper we present some architectures to deal with fast convolution computation based on carry save adders which are intended to be specifically implemented on FPGAs. Carry–save adders are not frequent in FPGA implementations since FPGA has a fast carry propagation path. In this paper we prove that it is possible to use carry-save arithmetic in a efficient way on FPGA for convolution operation. We make use of the specific structure of the FPGA to design an optimized accumulator which is able to deal with carry-save additions as well as carry-propagate additions using the same hardware. This lead to an efficient combined CSA-CPA architecture with fast computation and optimizing the hardware cost. Experimental results for different word lengths are presented to validate our proposal.**

I. Introduction

The convolution function is the key of a lot of digital signal processing applications (filtering, fft, correlation, neuronal networks ...). It is based on multiplication and accumulation operations. Due to its fundamental role in DSP applications, many high performance FPGA devices have incorporated special cells for DSP which deal with these operations. Nevertheless, it is not found in low cost FPGA devices.

To reduce the cost of the multiplication involved in applications with constant terms, a constant multiplier or distributed arithmetic is usually implemented. Nevertheless, in other applications the multiplication can not be avoided (adaptive filters, correlation, neuronal networks, ...). This has taken to manufacturers to use embedded multipliers to perform this kind of operations efficiently.

Convolution involves a lot of multiplications and accumulations. If an FPGA device has few multipliers, it is frequent to use them in an iterative way to implement the function. To increase the throughput it is necessary to reduce the cycle time specially when there are a lot of operations. Since the multiplier size is fixed in a FPGA device and the accumulator is customized by the user, a good design of the accumulator can lead to better performance of the full operation.

Many authors have implemented different design to carry out the convolution operation. Modern FPGA devices include fast carry-logic which allows the implementation of fast carry propagate adders. Thus, these designs use carry propagate adders [1], [2], [3]. In this paper we propose the use of carry-free adders as an efficient alternative which reduces the computation time of the convolution. It is based on a low level assignment of inner resources of the slices, which gives an optimized performance in comparison with automatic assignment [4].

On the other hand, recently there is an increasing interest of the scientific community to design new algorithms which take advantage of the special FPGA inner architecture [5].

In this paper we present an iterative architecture for convolution computation which is faster than previous implementations. It is achieved by using redundant arithmetic. In spite of redundant arithmetic involves an increase of hardware, we have developed a technique which allows us to reuse the hardware resources to obtain the final result in conventional arithmetic.

II. Architecture for Convolution

The architecture that we propose for convolution is based on an iterative use of the embedded multipliers presented in most of modern FPGA devices. In Fig. 1 a typical architecture for iterative multiplication and accumulation for convolution of two vectors is presented, where the operands are n-bits wide and the accumulator has $2n + i$ bits (we use i bits to prevent overflow in the additions). One value of each vector is introduced to the multiplier and its result is accumulated with the previous partial result on each iteration. Thus, the basic cycle time is $T_{cycle} = T_{mult} + T_{acc}$ and the time required for one computation is $T_{total} = N * T_{cycle}$ where N is the number of elements of the vectors.

To reduce the total computation time (T_{total}) we have to reduce the cycle time T_{cycle}. Multipliers are embedded (prefixed by the manufacturers) and can be pipelined to reduce its effective cycle time, whereas the accumulator can not be pipelined due to true dependence problems. To reduce the effective time of the accumulator we propose the use of carry–save adders (CSA). This arithmetic is specially useful when a lot of additions are carried out, which is the case of the convolution computation. Nevertheless, it requires a final conversion to conventional representation which is normally performed by

978-1-4244-5091-6/09/$25.00 ©2009 IEEE

Convolution Computation in FPGA Based on Carry-Save Adders and Circular Buffers

Carlos D. Moreno[1,4], Pilar Martínez[2,4], Francisco J. Bellido[1], Javier Hormigo[3,5], Manuel A. Ortiz[1], and Francisco J. Quiles[1]

[1] Computer Architecture, Electronics and Electronic Technology Department, University of Córdoba, Spain
[2] Applied Physics Department, EPS, University of Córdoba, Spain
[3] Computer Architecture Department, University of Málaga, Spain
[4] Campus Universitario de Rabanales. 14071 Córdoba, Spain
[5] Campus Universitario de Teatinos. 29080 Málaga, Spain
ellmomoc@uco.es

Abstract. In this article, we present some architectures to carry out the convolution computation based on carry–save adders and circular buffers implemented on FPGAs. Carry-save adders are not frequent in the implementation in FPGA devices, since these have a fast carry propagation path. We make use of the specific structure of the FPGA to design an optimized accumulator which is able to deal with carry–save additions as well as carry–propagate additions using the same hardware. On the other hand, this structure of circular buffers allows the convolution computation of two signals with two algorithms of calculation: the input side algorithm and the output side algorithm, in a more efficient way.

Keywords: Convolution, FPGA, carry-save adders, circular buffers.

1 Introduction

There are several methods to describe the relation between the input and output of the linear time invariant systems (LTI), when both are represented according to time. One of the methods to describe it is by means of differential linear equations (in continuous time) or equations in differences of constant coefficients (in discrete time). Another way of representing this relation would be by means of a block diagram which represents the system as an interconnection of three elementary operations: multiplication, addition, and displacement in the time for systems in discreet time, or integration for systems in continuous time. The third form is the description by means of variables of state, which corresponds to a series of differential equations or in differences of the first order connected that represent the behavior of the 'state' of the system and an equation that relates the state to the output.

However, the most widely used method to represent this relation is related to its response to impulse. The response to impulse is the output of the system associated with the input of the impulse. Considering the response to the impulse, we determine

M.L. Reyes et al. (Eds.): IT Revolutions 2011, LNICST 82, pp. 237–248, 2012.

UNIVERSIDAD D CÓRDOBA

idep

ED-UCO
Escuela Multidisciplinar de Doctorado-UCO

El Vicerrector de Estudios de Posgrado y Formación Continua y Coordinador de la Escuela Multidisciplinar de Doctorado de la Universidad de Córdoba

ACREDITA que:

CARLOS DIEGO MORENO MORENO

ha presentado la **comunicación oral** que lleva por título :

"COMPARACIÓN DE LA OPERACIÓN DE CONVOLUCIÓN EN DIFERENTES DISPOSITIVOS ELECTRÓNICOS DE BAJO COSTE"

en el **III Congreso Científico de Investigadores en Formación de la Universidad de Córdoba** celebrado los días 9 y 10 de abril de 2013.

Y para que así conste, se expide y firma este certificado en Córdoba, a 10 de abril de 2013

Fdo: JOSE CARLOS GÓMEZ VILLAMANDOS
Vicerrector de Estudios de Posgrado y Formación Continua
Coordinador de la Escuela Multidisciplinar de Doctorado ED-UCO

Rectorado de la Universidad de Córdoba – Avd. Medina Azahara 5, 14071 CÓRDOBA

Acelerador Hardware de bajo coste para bus PCI Convencional

Francisco J. Quiles, Manuel Ortiz, Miguel A. Montijano, Carlos D. Moreno, Maria Brox
Dept. Arquitectura de Computadores
Universidad de Córdoba
Campus Universitario de Rabanales
14071 Córdoba. Spain
email: ellorlom@uco.es

Javier Hormigo, Julio Villalba
Dept. Arquitectura de Computadores
Universidad de Málaga
Campus de Teatinos
29080 Málaga. Spain
email: hormigo@ac.uma.es

Resumen—**En este trabajo se presenta un acelerador hardware de bajo coste para el bus PCI convencional. El acelerador está formada por dos partes: un interfaz al bus PCI y una FPGA para la implementación del algoritmo del acelerador. La simplicidad de este acelerador hardware permite que el diseñador dedique su esfuerzo al algoritmo que implementará en la FPGA mientras que el esfuerzo que dedica al interfaz PCI es mínimo ya que el interfaz al bus PCI se ha reducido al máximo. El bridge con el bus PCI convencional está implementado en un core IP, que permite la programación dinámica de la FPGA e implementa facilidades para el control de este recurso por el sistema operativo.**

Palabras clave: acelerador hardware; FPGA; PCI Convencional; computación de alto rendimiento.

I. INTRODUCCION

En numerosas áreas de aplicación, especialmente en bioinformática, tratamiento digital de imagen y aplicaciones con alta precisión aritmética se requiere una alta demanda de procesamiento computacional (High Performance Computing). Esta necesidad está haciendo que emerjan nuevas tecnologías que exploten el enorme potencial de las FPGAs, gracias al paralelismo inherente en el hardware [1], a la vez que minimicen el esfuerzo requerido para llevar los algoritmos secuenciales a algoritmos paralelos e implementarlos en una FPGA.

Muchos trabajos de investigación están relacionados con la optimización de recursos en FPGAs y es en este campo donde se centra el trabajo de nuestro grupo. La optimización de recursos en FPGAs conduce a un menor coste económico y consumo de potencia del sistema. En este sentido los aceleradores hardware para HPC programables dinámicamente permiten cambiar el algoritmo programado en la FPGA y por tanto utilizar FPGAs de bajo coste. Por otro lado, otro objetivo que se persigue es conseguir que nuestros alumnos desarrollen prácticas y proyectos de alta tecnología donde se simplifica al máximo el sistema.

El resto del trabajo está organizado de la siguiente manera: en el capítulo 2 se presenta brevemente la plataforma UCORE desarrollada para utilización docente y de investigación para computación reconfigurable, en el capitulo 3 muestra la arquitectura general del acelerador hardware y en el capitulo 4 se muestra una de las placas que se ha desarrollado como componente hardware del acelerador y se mostrará la aplicación que maneja la configuración y programación del acelerador de forma interactiva.

II. PLATAFORMA PARA COMPUTACION RECONFIGURABLE UCORE

El acelerador hardware de bajo coste que se va a describir en este trabajo forma parte de la plataforma UCORE de computación reconfigurable que se presenta en [2]. La plataforma de computación reconfigurable UCORE se diseñó con propósitos educacionales y se está utilizando también como plataforma de prototipado rápido en trabajos de investigación. Los elementos que forman parte de la plataforma UCORE se muestran en la fig. 1, y consta de:

- La placa de prototipado rápido UCOS3E5 que contiene dos dispositivos programables, un CPLD y una FPGA y varios periféricos e interfaces más comunes. El CPLD realiza el interfaz al bus PCI convencional [3] y está conectado con la FPGA.
- IPs cores: El usuario dispone actualmente de varios IP cores para utilización con la placa de rápido prototipado o para cualquier otro sistema. Cabe destacar tres IP cores optimizados para el CPLD que actúa de puente entre el bus PCI convencional y la FPGA.
- Un analizador lógico virtual sencillo optimizado en la utilización de recursos (en desarrollo) que se integra en el sistema reconfigurable para su depuración. El analizador lógico tiene dos partes: un IP core que se integra con el diseño a depurar y extrae los datos del sistema y una aplicación en el ordenador personal que configura el disparo y muestra los datos.
- La aplicación, *PCI Bus Handler*, que permite el acceso al espacio de configuración de cada uno de los dispositivos PCI. En el caso de nuestro sistema reconfigurable nos permitirá localizar el dispositivo acceder al espacio de configuración y acceder al dispositivo en sí.

2010 International Conference on Reconfigurable Computing

UCORE: RECONFIGURABLE PLATFORM FOR EDUCATIONAL PURPOSES

Francisco J. Quiles, Manuel Ortiz, Maria Brox, Carlos D. Moreno
Department of Computer Architecture
University of Cordoba
University Campus of Rabanales
14071 Cordoba. Spain
email: el1orlom@uco.es

Javier Hormigo, Julio Villalba
Department of Computer Architecture
University of Malaga
University Campus of Teatinos
29080 Malaga. Spain
email: hormigo@ac.uma.es

***Abstract*— This paper presents an educational platform for digital system practices with a conventional PCI bus interface, based on reconfigurable hardware especially useful for the designing of hardware accelerators and systems with a PCI bus interface. The aim of the platform is to provide students with a single tool to develop rapid prototypes that covers all aspects involved in the study of digital systems. The platform consists of hardware and software components that allow to easily develop prototypes of general electronic systems, simple systems with a conventional PCI bus interface, hardware accelerators, and buses. The platform is focused on reconfigurable computing practices for university degrees in Electronics Industrial Engineering and Computer Engineering of the new framework of European Higher Education Area (EHEA).**

Keywords-Reconfigurable Systems; CPLD; FPGA; European Higher Education Area

I. INTRODUCTION

The development of Digital Electronic Systems has undergone a rapid evolution throughout the last decades. The first systems were built with low integration digital circuits and a high number of components. As the level of integration of components began to increase, the system components were grouped into functional blocks that were implemented in complex programmable logic devices (PLD) that contained mainly logic control [1].

Due to the large development and integration of programmable logic devices, current systems tend to be completely implemented in a single chip (SOC), and may even contain a large number of processors (MPSoC).

Parallel to this development, the teaching of electronic systems has had to change. The programmable logic devices and reconfigurable systems are an important part of studies of Electronics Industrial Engineering and Computer Engineering [2]. In addition, in new degrees tailored to the European Higher Education Area (EHEA) [3], embedded systems and reconfigurable systems have become particularly relevant.

The rest of the paper is organized as follows: Chapter II will discuss the teaching methodology, Chapter III will show all the components of the UCORE platform used in PLDs practices of the current studies of Electronics Industrial Engineering and supplemented for the new EHEA studies. Chapter IV will describe the different types of practices that can be carried out with this platform, and finally conclusions and future work will be shown.

II. TEACHING METHODOLOGY

In our opinion (which is also widely believed), the fact that students have a basic education in digital systems is essential for the development of digital electronic systems, and therefore, it is compulsory in degrees in Electronics Engineering and Computer Engineer (either electronic or digital) [4]. This basic education begins with the study and development of systems with logic gates and flip-flops; following their basic education with the analysis and design of small functional blocks such as decoders, multiplexers, registers, and counters. To complete the basic education, the students learn the fundamentals of semiconductor memories and small units of control [5]. This is the ideal starting point in order for students to begin the study of programmable logic devices to complete the study and development of complex reconfigurable systems.

Given the horizontal nature of knowledge in EHEA studies, the students should cover the main stages of the development of a reconfigurable system. Regardless of the physical design of the electronic system, including the PCB, the students should:

1 Design the overall system- functional blocks level.
2 Identify external interfaces that the system will have.
3 Determine the functionalities that will be integrated into programmable logic devices.
4 Design features of the programmable logic device-architecture and functional blocks level, and make the description in a high-level language, such as VHDL.
5 Simulate the design that will be integrated into the programmable logic device with a simulation tool such as ModelSim.
6 Programme programmable logic devices.
7 Debug the reconfigurable system.
8 Check the external interfaces.
9 Carry out a global testing.

The teaching methodology used in degrees at the University of Cordoba and Malaga is the one shown in Figure 1. The first part is carried out with commercial synthesis tools.

978-0-7695-4314-7/10 $26.00 © 2010 IEEE
DOI 10.1109/ReConFig.2010.60

Using Soft Processors for Component Design in SOC: A Case-Study of Timers

M. Ortiz, M. Brox, F. Quiles, A. Gersnoviez, C. Moreno, M. Montijano

Universidad de Córdoba. Departamento de Arquitectura de Computadores, Electrónica y Tecnología Electrónica
Edificio Leonardo Da Vinci. Campus Universitario de Rabanales, 14071-Córdoba, Spain
el1orlom@uco.es

Abstract—**System on Chip (SOC) could be considered as a very useful alternative in the design of real-time systems, especially due to the possibility of integrating several processors in just one FPGA. This strategy enables the use of soft processors to design the system's components, which have traditionally been developed by hardware. In this paper we study a HW/SW co-design of a timer pool for its use in SOC, which is constructed by a Picoblaze soft processor. Our approach offers a novel alternative among hardware and software timers that increases the overall system performance, and achieves a higher precision than software timers with a considerable reduction in cost and area occupied.**

I. INTRODUCTION

Timers play an important role in any scheduler for real-time systems. In this case, the scheduler must be carefully designed in order to have few overheads in the system, especially in timer management. Some authors implement the scheduler in hardware to address the overhead [1, 2, 3]. On the other hand, applications related to periodic data acquisition, motor control, signal generation, pulse counting and loop timeout use a great number of timers.

The use of soft processors can be an alternative in hardware-software co-design. This alternative is explored in [4] where three different scheduler implementations are investigated. A software implementation uses a processor to run the scheduler and the application tasks. A software-software implementation uses a processor to run the application tasks, and a co-processor to run the scheduler. In a hardware-software implementation, the scheduler is implemented directly in the hardware.

Different timer resolutions are required depending on the application. When a low precision is required, a software solution is a good option. On the other hand, if a high precision is demanded, a hardware implementation is necessary.

When a system is considered, it is analysed the number of hardware and software timers to implement depending on the cost and processing capacity respectively. Later on in this paper, we are going to perform a study of HW and SW timers showing a very interesting alternative, especially for SOC, by using a simple soft processor.

II. HARDWARE TIMERS

Hardware timers are timers implemented by hardware circuit logic. They are counters that work to a fixed frequency. They have a high precision and do not create an overhead for the processor. Implementing hardware timers is the most efficient way to achieve timers in a computing system. This is the most accurate option and the one that involves the lowest overhead in the system. However, the implementation cost and the area occupied on the FPGA is high.

Theoretically the number of HW timers could be unlimited. However the occupied size on the FPGA, the power consumption and programming complexity impose some limitations in SOC. Because of this, it is not usual to implement a high number of hardware timers in systems. For instance, a circuit with sixteen autoload timers of sixteen bits implemented into a Xilinx Spartan-xc3s200 consumes 496 Slices of the FPGA, approximately 12% of the available resources, without including the external interface logic.

III. SOFTWARE TIMERS

Hardware timers are limited in a system. Because of this, it is necessary to use software timers for applications that require an unlimited number of timers.

Software timers are a piece of code connected to the system clock interrupt. Each timer is represented by a data structure. A basic data structure is shown in Fig. 1.

```
struct timer {
        int used; /* TRUE if in use */
        TIME time; /* time left */
        TIME period; /* time to wait */
        int *event_timeout; /* set to TRUE at timeout */
} timers[MAX_TIMERS]; /* set of timers */
```

Fig. 1. Basic data structure of timer.

1-4244-2542-6/08/$20.00 ©2008 IEEE